Control Inteligente De Semáforos Mediante Aprendizaje Automático

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Sergio Guerrero Paredes.

Tutora: Raquel Fuentetaja Pizán.

Fecha: 25/09/2016

ESCUELA POLITÉCNICA SUPERIOR. UNIVERSIDAD CARLOS III DE MADRID.

Índice de contenido

Resumen 5
Abstract 6
1. Introducción 16
1.1. Estructura del documento 16
2. Motivación y objetivos 18
3. Estado del arte 20
3.1. Aprendizaje automático 20
3.1.1. Aprendizaje por refuerzo 20
3.1.2. Otras técnicas de aprendizaje automático 23
3.2. Control automático de semáforos 24
3.3. SUMO 26
3.4. Python 28
3.5. Open Street Map 28
3.6. JOSM 30
4. Gestión del proyecto 32
4.1. Metodología 32
4.2. Planificación 35
4.3. Presupuesto 36
4.3.1. Coste del personal 37
4.3.2. Coste de los bienes tangibles 37
4.3.3. Costes indirectos 39
4.3.4. Precio total 39
5. Descripción del trabajo realizado 41
5.1. Requisitos software 41
5.1.1. Requisitos funcionales 42
5.1.2. Requisitos no funcionales 46
5.2. Diseño del software 48
5.3. Descripción del espacio de estados 52
5.4. Descripción del espacio de acciones 54
5.5. Descripción de la función de refuerzo 54
5.6. Descripción de los parámetros del Q-Learning 55
6. Experimentación y resultados 57
6.1. Descripción de los periodos de aprendizaje 57

6.1.1. Generación de tráfico 57
6.2. Descripción de los escenarios 58
6.2.1. Escenario 1 – Mapa real pequeño con poco tráfico 58
6.2.2. Escenario 2 – Mapa real pequeño con mucho tráfico 60
6.2.3. Escenario 3 – Cuadrícula mediana con poco tráfico 60
6.2.4. Escenario 4 – Cuadrícula mediana con mucho tráfico 62
6.2.5. Escenario 5 – Cuadrícula grande con poco tráfico 62
6.2.6. Escenario 6 – Cuadrícula grande con mucho tráfico 63
6.3. Evaluación de los resultados 64
6.3.1. Resultados Escenario 1 64
6.3.2. Resultados Escenario 2 67
6.3.3. Resultados Escenario 3 70
6.3.4. Resultados Escenario 4 73
6.3.5. Resultados Escenario 5 76
6.3.6. Resultados Escenario 6 79
7. Conclusiones y líneas futuras 83
Marco legal 85
Bibliografía 86

Índice de tablas

Tabla 1: Coste del personal 37
Tabla 2: Coste de los materiales 38
Tabla 3: Amortización de los materiales 39
Tabla 4: Alquiler del inmueble 39
Tabla 5: Coste total del proyecto 40
Tabla 6: Plantilla requisitos 41
Tabla 7: Requisito funcional 1 42
Tabla 8: Requisito funcional 2 43
Tabla 9: Requisito funcional 3 43
Tabla 10: Requisito funcional 4 44
Tabla 11: Requisito funcional 5 44
Tabla 12: Requisito funcional 6 45
Tabla 13: Requisito funcional 7 45
Tabla 14: Requisito funcional 8 46
Tabla 15: Requisito no funcional 1 46
Tabla 16: Requisito no funcional 2 47
Tabla 17: Requisito no funcional 3 47

Tabla 18: Requisito no funcional 4 48
Tabla 19: Escenario 1: Resultados 67
Tabla 20: Escenario 2: Resultados 70
Tabla 21: Escenario 3: Resultados 73
Tabla 22: Escenario 4: Resultados 76
Tabla 23: Escenario 5: Resultados 79
Tabla 24: Escenario 6: Resultados 82

Índice de ilustraciones

Illustration 1: Basic reinforcement learning model. Downloaded from [2] 7
Illustration 2: System arquitecture 11
Illustration 3: TraCI-SUMO connection. Downloaded from [14] 12
Illustration 4: TraCI-SUMO disconnection. Downloaded from [14] 12
Ilustración 5: Modelo estándar de aprendizaje por refuerzo. Extraído de [2] 21
Ilustración 6: Comparativa de sistemas inteligentes de tráfico. Extraído de [7] 25
Ilustración 7: Resultados del sistema SURTRAC. Extraído de [8] 25
Ilustración 8: Interfaz de SUMO 27
Ilustración 9: Interfaz de OpenStreetMap 29
Ilustración 10: Exportar mapa en OpenStreetMap 29
Ilustración 11: Interfaz de JOSM 30
Ilustración 12: Modelo en cascada 33
Ilustración 13: Planificación del proyecto 36
Ilustración 14: Arquitectura del sistema 49
Ilustración 15: Conexión TraCI-SUMO. Extraído de [14] 50
Ilustración 16: Desconexión TraCI-SUMO. Extraído de [14] 51
Ilustración 17: Mapa real pequeño 59
Ilustración 18: Cuadrícula mediana 61
Ilustración 19: Cuadrícula grande 63
Ilustración 20: Escenario 1: Aprendizaje 65
Ilustración 21: Escenario 1: Q-learning / Política fija 66
Ilustración 22: Escenario 1: Política aleatoria 66
Ilustración 23: Escenario 2: Aprendizaje 68
Ilustración 24: Escenario 2: Q-learning / Política fija 69
Ilustración 25: Escenario 2: Política aleatoria 69
Ilustración 26: Escenario 3: Aprendizaje 71
Ilustración 27: Escenario 3: Q-learning / Política fija 72
Ilustración 28: Escenario 3: Política aleatoria 72
Ilustración 29: Escenario 4: Aprendizaje 74
Ilustración 30: Escenario 4: Q-learning / Política fija 75
Ilustración 31: Escenario 4: Política aleatoria 75
Ilustración 32: Escenario 5: Aprendizaje 77
Ilustración 33: Escenario 5: Q-learning / Politica fija 78
Ilustración 34: Escenario 5: Política aleatoria 78

Ilustración 35: Escenario 6: Aprendizaje 80
Ilustración 36: Escenario 6: Q-learning / Política fija 80
Ilustración 37: Escenario 6: Política aleatoria 81

Índice de ecuaciones

Equation 1: Q-table update formula 8
Ecuación 2: Actualización de la tabla Q..... 22
Ecuación 3: Función de refuerzo 55

Resumen

Este trabajo se centra en el control de semáforos de forma inteligente. Se presentan varios sistemas funcionales en la actualidad que utilizan diversas técnicas de Inteligencia Artificial para optimizar la gestión de tráfico, desde árboles de decisión y redes de neuronas hasta algoritmos genéticos.

Otra técnica de la Inteligencia Artificial altamente expandida en este campo es el aprendizaje por refuerzo. Gracias a esta técnica, los semáforos son entrenados inmersos en un entorno específico de tráfico y son capaces de aprender a través de la experiencia.

En este proyecto se presenta un sistema multi-agente distribuido, donde se aplica aprendizaje por refuerzo (Q-learning) para resolver el problema de la gestión de tráfico. La particularidad de este sistema reside en que cada agente controla solamente un semáforo, basándose en la información local del mismo, como el número de vehículos o la velocidad media en su carril.

Este trabajo se desarrolla en el simulador SUMO. El sistema interactúa con SUMO a través de la interfaz TraCI, que ofrece todas las herramientas necesarias para controlar la simulación a la vez que pone a disposición del usuario todas las métricas mencionadas anteriormente.

Para evaluar el sistema desarrollado se ejecuta en varios escenarios, un sistema de política fija y un sistema de política aleatoria. Cada escenario dispone de un mapa y un nivel de tráfico distinto, con el objetivo de comprobar el comportamiento de cada sistema con tráfico fluido y congestionado. En base a los resultados que estos experimentos arrojan, se concluye que el sistema desarrollado gestiona de forma más eficiente el tráfico congestionado que el de política fija, aunque es menos eficiente en situaciones de tráfico fluido.

Abstract

This project is about creating a system that manages traffic efficiently. Big cities have some serious troubles when it comes to traffic jams and this has a big impact on both the economy and the enviroment.

The study *The future economic and environmental costs of gridlock in 2030* [1] made by the CEBR (Centre for Economics and Business Research) concludes that the total cost of traffic jams in just four big countries (France, Germany, Great Britain and the United States) raises up to 200 billion dolars in 2013. But this can be lower if an intelligent system is created and implemented.

The current system that is installed all around the world is a centralized system that contains traffic lights that only changes in a fixed interval. This system has the exact same behavior regardless of the situation of the traffic and it fails at avoiding traffic jams.

In the recent past, some researchers have created intelligent traffic lights that make decisions based on the traffic that exists in that exact moment, improving the traffic flow and reducing the jams. But these systems are generally centralized, which means they are not scalable.

In this project, we will try to create a distributed multi-agent system using machine learning, where each traffic light will be independent and will only have information about the lane they are controlling. This way, each traffic light will learn to make better decisions and the system will be highly scalable due to the possibility of installing it anywhere, regardless of where it is situated.

This system will be integrated in the simulator chosen to do the tests. This simulator is **SUMO**, a traffic simulator that offers a realistic vehicle behavior and an easy interface called TraCI. This interface allow us to control the simulation, change any traffic light at will and get any information about number of vehicles or their speed in any lane. It also supports many programming languages: Python, Java, Matlab and C++. It has some disadvantages like the inability to rewind time, but it is the best simulator to use in this case.

As it was said previously, **machine learning** will be used in the system that it is going to be created. Machine learning is a subfield in computer science which objective is to develop some techniques to give computers the ability to learn. These techniques are about generating behaviours from examples and they focus on solving classification, optimizations and decision-making problems. Some of its applications are: detecting credit card fraud, speech and handwriting recognition, robot locomotion and medical diagnosis.

Reinforcement learning is an area of machine learning that it is based on a software agent that must learn a behavior through trial and error, interacting with an enviroment so as to maximize a reward. The reinforcement model consists of:

- A set of states representing the enviroment at a given time.
- A set of actions that the agent can perform.
- A reward function that defines the reward received by the agent after performing an action.

The interaction between the agent and the enviroment is represented in the illustration 1:

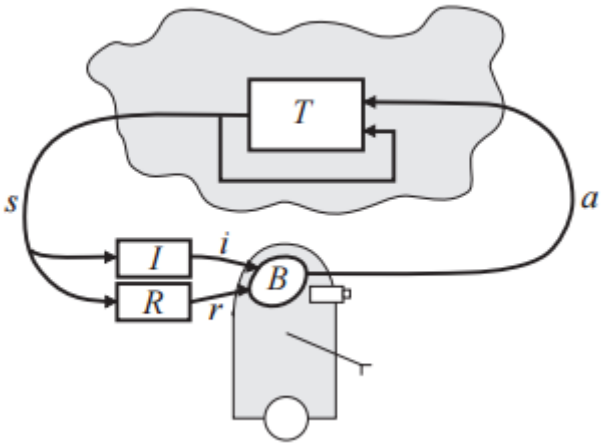

Illustration 1: Basic reinforcement learning model. Downloaded from [2]

The agent (named 'B' in the image) performs an action 'a' in the enviroment 'T'. The state 's' gets information about the enviroment and calculates the reward. This process is called cycle.

The agent uses the information gathered during a set of cycles to learn a behavior, establishing what actions are beneficial in which situations.

Each of these cycles is represented by a tuple. A tuple contains the information about the state in the time 't', the action that was performed by the agent in that time, the state in the time 't+1' and the reward. This cycle is repeated several times so the agent can explore different behavior in order to come upon a an optimal behavior.

The reward will depend on the behavior that we are trying to achieve with the agent. Typically, this reward will be included in an interval between a negative value and a positive one, approaching to the minimum value when the actions taken by the agent are not beneficial and approaching to the máximum value when they are.

Additionally, there are some strategies that affects the decision of the agent when it comes to which action to perform. This strategies are called exploration and explotation policies, and one of these is **ε-greedy**. It defines a probability of ε to choose the best action and a probability of (1 - ε) to choose a random one. The value of ε is reduced throughout the learning phase until it gets to zero. This way, the agent will explore more states in the beginning of the learning process and it will end up performing only the best actions at the end. This method is used in other similar projects with good results, so it will be used in this project aswell.

Q-learning is a reinforcement learning technique that was first introduced by Watkins in 1989. This technique gives the agent the ability to learn and act optimally in Markov decision processes. The agent performs actions and receives a reward. This way, it is not required to describe the whole domain as the agent will explore it in order to optimize its behavior.

The algorithm consists of: a discrete set of states, a discrete set of actions, a reward function, a table where the learning results are stored and a function that defines how this table is updated.

The states, actions and reward function are defined by the problem domain. The design of these elements are the most important part of the system and this will have a heavy impact on the final result. Later on, these elements are going to be described.

But, first, let's take a look at how the table is updated. This table is named Q-table and it is updated from the information contained in the tuples that were described before. This update is performed as shown in the equation 1:

Equation 1: Q-table update formula

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot {}^{max}_{a}\mathrm{Q}(s_{t+1}, a) - Q(s_t, a_t)\right)$$

Where each one of the elements that integrates the equation are described as follows:

- Q (s_t, a_t): Value of the Q-table for the state 's' and the action 'a' at the time 't'.
- α: Learning rate. This rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information.
- r: Reward received after performing the action 'a' in the state 's' at the time 't'.
- γ: Discount factor. This rate determines the importance of future rewards. A factor of 0 will make the agent consider only recent rewards, while a factor approaching 1 will make it strive for a long-term high reward.
- ${}^{max}_{a}\mathrm{Q}(s_{t+1}, a)$: Maximum value for state 's' in the Q-table.

The Q-table is updated based on the previous experience, the reward received at a certain time and an estimation of the reward that the agent will get in the future. This is controlled by the learning rate and the discount factor. In this project, after a testing process where these rates changed from one test to another, we found that a good value for the learning rate was 0.7, while the discount factor was 0.2. As there is no method to find the optimal value for these parameters and the available time is limited, no fine tuning is going to be addressed.

To sum up, the Q-table will have as many rows and columns as states and actions are defined. The agent will perform different actions depending on the information stored in this table. And during the learning process, the ε-greedy policy will determine the action to be taken, whether it is the optimal one or a random one.

The states, actions and the reward function are going to be described now. The attributes of the **state** have to represent faithfully the enviroment at any given time. From all the metrics available in SUMO, only three of them were chosen to describe the enviroment:

The traffic light {Red, Green}. The value of this parameter will be Red or Green depending on the status of the traffic light of the lane that is being represented.

The number of vehicles {Very low, Low, Medium, High}. Very low represents that there are no vehicles in the lane; if there are one or two vehicles, the value will be Low; Medium if there are three, four or five; and high if there is six or more vehicles in the lane.

The mean speed {Low, Medium, High}. This attribute will take the value Low when the mean speed of the lane is less than 20% of the maximum speed of that lane; Medium when this percentage is between 20% and 70%; and High when this is higher than 70%.

On the other hand, there are only two **actions** possible: changing the traffic light and not changing it. To perform this action, the current status of the traffic light is needed (in order to change it to Green if it is Red and vice versa). But this information is currently in the state, so there is no trouble in this matter.

Finally, the **reward function**. This function will determine the reward that the agent gets for each action taken. As the objective of this system is to improve the traffic flow, the reward has to reflect how the action has affect it. It has been considered that the best metric available to measure this is the waiting time of all the vehicles in a lane. When the waiting time is zero, the reward will be one. When the waiting time increases, the reward decreases until it gets to minus one.

Besides the technique used in this project, there are several other techniques used in similar projects to address the problem of traffic management with intelligent systems. For example, in the paper *ETC assisted traffic light control scheme [4]*, **decision trees** are used to model the behavior of the traffic lights. **Neural networks** is the technique used in the system described in the paper *Neural Networks for Real-Time Traffic Signal Control* [5]. And in the paper *Traffic Control with Standard Genetic Algorithm* [6] we can see another example, this time using **genetic algorithms**.

Focusing on this project, a methodology is still needed to be defined. A methodology serves to plan the development of the system, dividing it in several phases so it is easier to reach the objectives of the project.

The methodology chosen in this case is the **waterfall model**. It gets this name because it suggests the image of a waterfall as the project phases flow down towards, each phase starts at the end of the previous one.

The phases defined in this model are: Requirements analysis, System design, Implementation, Verification and Maintenance.

The main disadvantage of this model is the rigidity of it. If you have to redo any phase, you will have to redo all the previous ones first. So any small change in any of the late phases would mean a lot of work.

Despite this, this methodology is easy to follow and I am familiar with it, that is why it was the one chosen.

Once the methodology is defined, the next thing to do is the **planning** of the project. This divides the entirety of the project in several phases, determining the start date and the end date for each one of them. This is crucial to reach the objectives in a short period of time, and it is done by giving each phase only the time that it is needed to be completed.

The illustration number 13 shows the planning of the project by weeks, indicating with a grey color the weeks given to each phase.

Next **the budget of the project** is shown. The final price of the project is 19.791,82 €, taxes included. This consists of the personal costs, the tangible goods and the indirect costs. This price is valid for a period of three months since it is displayed to the client. Further details of the budget are presented in the chapter called 'Presupuesto', although this has not been translated to english.

The next phase according to the planning made is the **requirements** analysis. The software requirements references different characteristics of the system: what the system will be able to do, its ease of use, its availability and some other restrictions. These requirements are going to be divided in two: functional requirements and nonfunctional requirements.

The functional requirements are the services that the system must provide. These requirements complements the nonfunctional requirements, which focus on the design and the implementation of the system.

The nonfunctional requirements indicates how the system is supposed to be build. These consists of the following aspects of the system: the performance, the availability, the ease of use and the scalability.

Further details on the requirements defined for this project can be found in the chapter called 'Requisitos software', although this has not been translated to english.

The next phase according to the planning is the **system design**. A description of how the system has been designed, the components it has and what is the function of each one of them can be found below.

The system presents a layered design pattern, where each layer can communicate with the one above it and below it, as the illustration 2 shows.

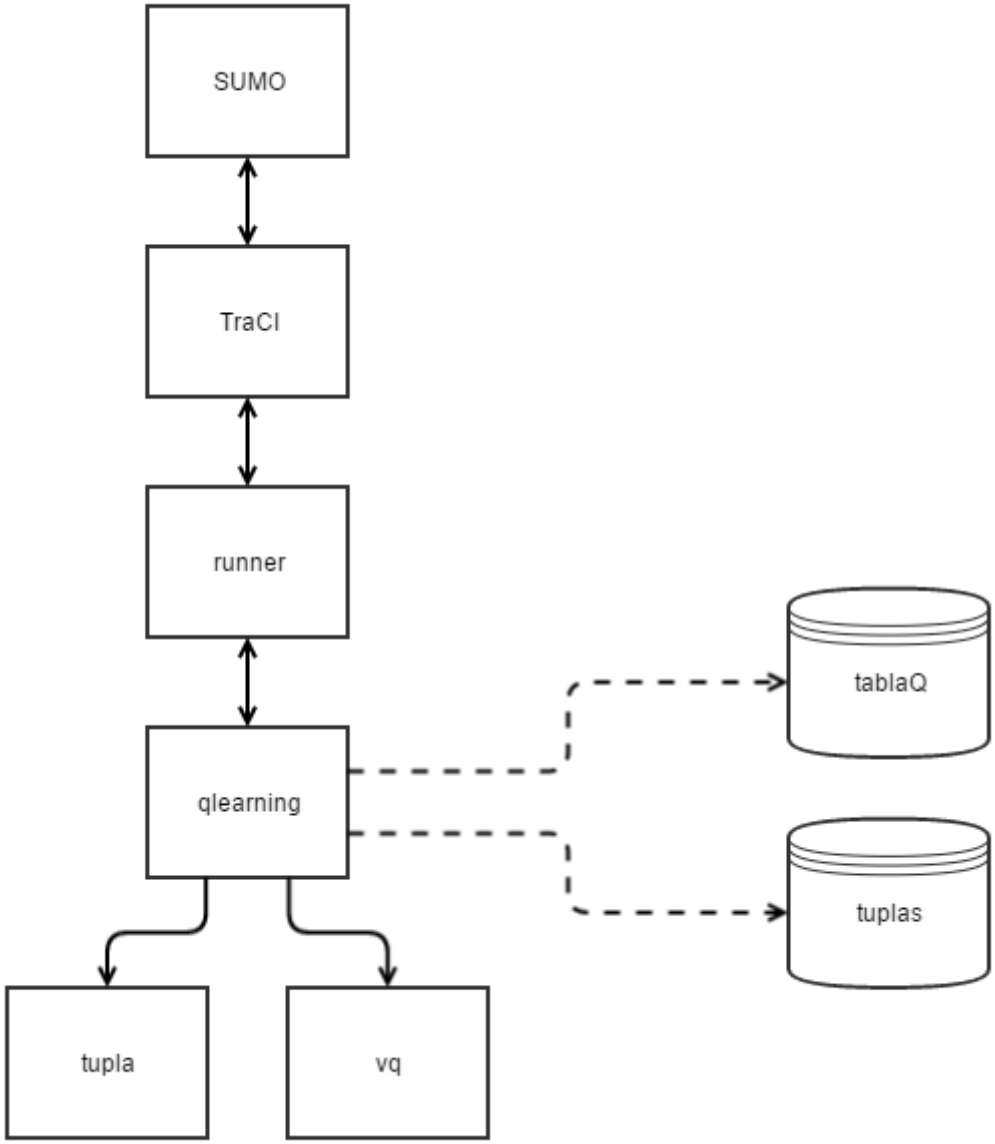

Illustration 2: System arquitecture

The first layer is **SUMO**, the traffic simulator used in this project. This is a subprocess that runs as a server during the simulations. It is the one that manages the GUI and the simulations, and it opens a port (8873) so the user can communicate with the simulator through this port.

To communicate with the SUMO server, the tool **TraCI** is used. This tool connects with the server from the client process as it is shown in the illustration 3.

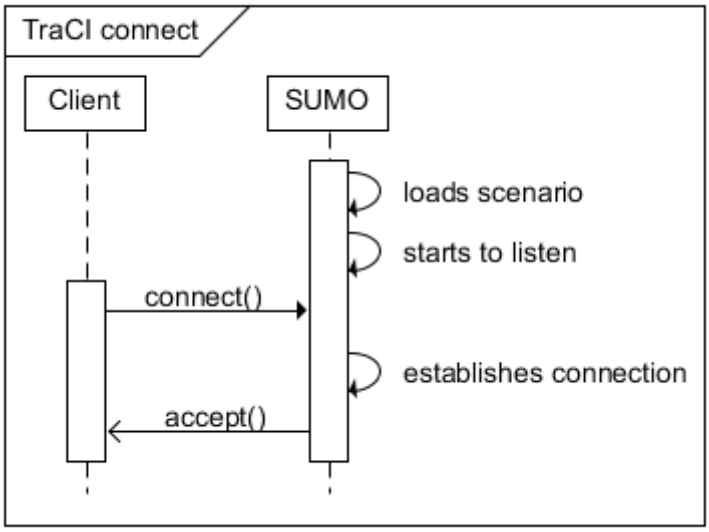

Illustration 3: TraCI-SUMO connection. Downloaded from [14]

Once the client is connected to SUMO, TraCI offers total control over the simulator. Thanks to this tool, we can start the simulation, stop it, get the state of every traffic light and modify it, get the maximum speed of a lane, get the number of vehicles in a lane, get the number of stopped vehicles in a traffic light and the waiting time of these vehicles, among many other functions.

When the simulation is finished, TraCI closes the communication with SUMO as displayed in the illustration 4.

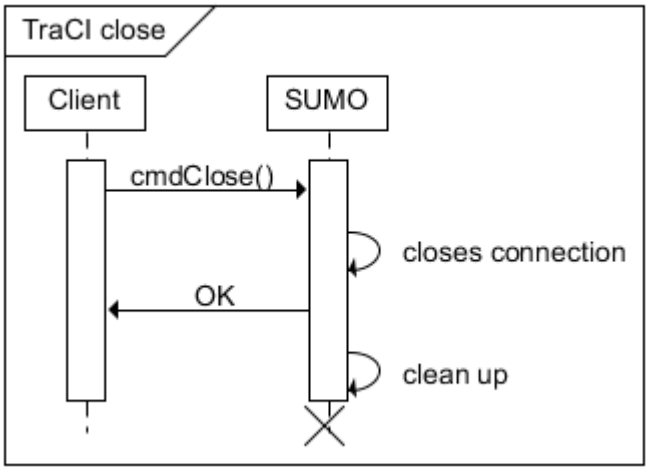

Illustration 4: TraCI-SUMO disconnection. Downloaded from [14]

The next component in the system is called **runner**. This component manages the simulation and the learning process. It communicates with TraCI to obtain all the necessary information from the simulator and performs a preprocessing on the data before passing it to the following components.

Qlearning is the component where the actual learning is performed. It gets from the upper layers the information it needs to update the Q-table. It also has some functions to choose between the best action always or an ε-greedy policy.

It is also possible to save the information gathered from the upper layers (the tuples) and the Q-table in the hard disk, so this information is available in future simulations. The system can perform offline learning with this information aswell.

The component **tuplas** is a class that contains the information about the tuples and some functions to Access this information (current state, next state, action and reward).

Lastly, the component **vq** (vector quantization) offers the possibility of quantifying the states and the actions. This means that, given any state or action, it determines to which state or action of the model it belongs. This is a highly reusable component and it is used mostly in clustering algorithms.

The layered design offers an easy way to accomplish all the requirements established in the previous phase, besides making the implementation of the system easier and increasing the reusability of pretty much all the components in future projects.

The next process to talk about is the **experimentation**. The goal of this process is to test the system under certain circumstances and compare the results with the results of other systems. But first, the details of the learning episodes are going to be described.

Each learning episode will be 10.000 simulation cycles long, which is about 3 hours of real traffic. After each cycle, the value of the learning rate is reduced and the value of ε is incremented. This way, the model is going to suffer big changes at the beginning of the episode and minor changes at the end.

The learning episode will be performed as follows: applying machine learning in a single traffic lights, while the rest of them follow a fixed policy. When the learning process is over, the Q-table is saved in disk and another episode is executed, applying this time the Q-table generated in all the traffic lights.

During the learning episode, local metrics will be taken to check the evolution of the system in this phase. The metric to use will be the accumulated waiting time for all the vehicles in the lane of the traffic light that is performing the learning process. This way, the improvement in the decision making of the traffic light can be seen.

During the evaluation episode, global metrics will be taken to compare the results of this system with a system that follows a fixed policy and a system that follows a random policy. The metric in this case will be the mean waiting time in all the traffic lights of the system.

To generate the traffic for each experiment, the tool *Random Trips* [16], which is part of the SUMO software, is used. This tool offers a **random generated traffic** for any map. The traffic density and other attributes like the speed of the cars at the start of the route can be modified through parameters, providing to the user the ability to generate different types of traffic.

To test the system behavior in different situations, a series of **experiments** are going to be defined. There are going to be six different experiments:

- The first one will be a small map with a low traffic density, generating only one vehicle for each cycle and a half. This map was downloaded from OpenStreetMap and it represents a centric area of Madrid city. It can be seen in the illustration number 17. The main purpose of this experiment is to test the behavior of the system under not congested traffic situations.
- The second one will have the same map as the first experiment but a higher traffic density, generating one vehicle for each 0.75 cycles. The main purpose of this experiment is to test the behavior of the system under congested traffic situations.
- The third one will have a bigger map than the first and second experiments, with a low traffic density, generating one vehicle each cycle. The shape of this map resembles the shape of big cities like Barcelona or New York. This map can be seen in the illustration number 18. The main purpose of this experiment is to test the behavior of the system under not congested traffic situations in a medium scene.
- The fourth one will have the same map as the third experiment but a higher traffic density, generating one vehicle each 0.85 cycles. The main purpose of this experiment is to test the behavior of the system under congested traffic situations in a medium scene.
- The fifth one will have a bigger map than the third and fourth experiments, with a low traffic density, generationg one vehicle each 0.65 cycles. This map can be seen in the illustration number 19. The main purpose of this experiment is to test the behavior of the system under not congested traffic situations in a big scene.
- The sixth and last experiment will have the same map as the fifth one but a higher traffic density, generating one vehicle each 0.35 cycles. The main purpose of this experiment is to test the behavior of the system under congested traffic situations in a big scene.

After analyzing the system behavior in all these experiments and comparing it to systems that follows a fixed policy and a random policy, the results throw the following conclusion: the system created gets better results than the fixed policy in the experiments number 2, 4 and 6, which are the ones that have a high traffic density. Meanwhile, the system with the fixed policy gets better results in the other three experiments, where the traffic density was smaller.

The main objective of this project was to develop a system that was able to manage traffic efficiently and to reduce the traffic congestions, and the results shows that this objective was accomplished. However, the fixed policy system was better in not congested traffic situations. Because of this, it is recommended to keep improving this system before installing it in real locations.

To sum up, this projects presents the possibility of creating intelligent traffic lights that are independent, providing this way the capacity to implement a system highly scalable in any place.

Nonetheless, this system can be improved in a lot of aspects. The design of the states and the reward function, which are relatively simple, could be much more complex, with the purpose of improving the final behavior of the system.

On the other hand, the learning process depends on many parameters. These parameters could be modified to improve the results. This means lengthening or shortening the duration of the learning process, adjusting the learning rate and the discount factor of the Q-learning algorithm, and using an exploration and explotation policy different than the ε-greedy.

Another improvement could be applying machine learning to discretize the attributes of the states. In order to do this, a k-means could be used, making use of the module named vq. This process has a huge impact on the final result, so achieving a better discretization could lead to a much more efficient behavior.

The experimentation is another area that could be worked in more depth in the future. Pedestrians could be included in the simulations, which would affect the traffic aswell. The experiments could incorporate different types of crossing, creating asymmetrical maps with different traffic densities in different areas of the map.

A system that avoids accidents in the crossings could come in handy in the future. The system design makes it possible to create dangerous situations as each traffic light only has information about its lane, so it does not know the state of the other traffic lights in its crossing. Thereby, there could be situations where all the traffic lights are open, giving rise to accidents. For this reason, it is suggested the inclusion of a component in the system that regulates this situations, even though this could affect the scalability of the system.

These are just a few examples of how the developed system can be improved in the future. Other projects can take this as a basis, focusing on performing some of the tasks mentioned before, without worrying about the integration of Q-learning and SUMO.

1. Introducción

Muchas personas a lo largo y ancho del mundo se topan todos los días con atascos. Esto es especialmente cierto en las grandes ciudades, donde las congestiones de tráfico son un verdadero problema cuya solución aún no se ha encontrado.

Para manejar el tráfico y evitar estas saturaciones, existen los semáforos: aparatos automatizados que regulan el flujo de los vehículos mediante señales luminosas. Sin embargo, los semáforos actuales son sistemas centralizados con una política de cambio de señales fija. Esto significa que siempre se comportan igual, independientemente de la situación en la que se encuentre el tráfico.

En los últimos años, se han intentado crear semáforos inteligentes que tomen decisiones en base al flujo del tráfico que exista en ese momento, mejorando así la gestión de las congestiones. Estos sistemas son, en su mayoría, centralizados, lo cual significa que son difícilmente escalables. Además, algunos de ellos tienen problemas de seguridad, por lo que requieren constante supervisión humana para su correcto funcionamiento y esto hace aumentar la desconfianza de la gente hacia este tipo de sistemas.

En este trabajo se va a intentar crear un sistema multi-agente distribuido mediante aprendizaje automático, donde cada semáforo será independiente. De esta forma, cada semáforo aprenderá a tomar mejores decisiones. El sistema será altamente escalable ya que la lógica del sistema no está centralizada, si no que cada semáforo se adaptará a su situación de forma automática.

Para el aprendizaje automático de cada semáforo, se utilizará el algoritmo Q-learning en un entorno controlado de simulación donde poder hacer pruebas de forma segura. Para ello, se usará el simulador SUMO, el cual ofrece un comportamiento realista y está dotado de características que ayudarán al desarrollo de este sistema.

Así, se intentará mejorar el funcionamiento de los sistemas actuales de política fija, ofreciendo una gestión más eficiente del tráfico y reduciendo los atascos, a la vez que se ofrecerá la posibilidad de instalar el sistema en cualquier semáforo de cualquier ciudad.

1.1. Estructura del documento

Este documento se divide en varias secciones: *Introducción*, *Motivación y objetivos*, *Estado del arte*, *Gestión del proyecto*, *Descripción del trabajo realizado*, *Experimentación y resultados*, y, por último, *Conclusiones y líneas futuras*.

En la *Introducción* se describe brevemente de qué trata el proyecto, por qué es importante y se define el alcance del mismo.

En el apartado *Motivación y objetivos* se exponen las razones por las que se ha elegido hacer este proyecto, así como cuáles son las metas que se quieren alcanzar con la realización del mismo.

En el *Estado del arte* se especifican las herramientas utilizadas durante el desarrollo del trabajo, exponiendo sus ventajas e inconvenientes, y se realiza un pequeño estudio sobre otros proyectos llevados a cabo en esta misma área de investigación.

En la *Gestión del proyecto* se recoge toda la información relativa a la metodología elegida, la planificación del trabajo y el presupuesto previsto para el desarrollo del sistema.

En la sección *Descripción del trabajo realizado* se detalla el desarrollo del proyecto de principio a fin, desde el análisis de los requisitos y el diseño del sistema hasta la descripción de los estados, de las acciones, de la función de refuerzo y del algoritmo utilizado, así como el proceso para establecer los parámetros del aprendizaje.

En el apartado *Experimentación y resultados* se muestra el procedimiento seguido a la hora de establecer los periodos de aprendizaje, se describen los escenarios utilizados durante los mismos y se presentan los resultados obtenidos, comparándolos con los resultados de otros sistemas.

Por último, en las *Conclusiones y líneas futuras* se relacionan los resultados obtenidos con los objetivos planteados y se plantean posibles mejoras al proyecto realizado, con afán de que éstas sean realizadas en futuros trabajos que tomen éste como base.

Además, se dispone de un pequeño resumen en español del trabajo y de un resumen más extenso en inglés al principio del documento, así como de la bibliografía al final del mismo.

2. Motivación y objetivos

En el mundo civilizado actual, los vehículos y las carreteras forman parte fundamental de nuestras vidas. Tanto, que es imposible imaginarse cómo serían nuestras vidas sin estos inventos.

Como conductor o pasajero, todos hemos sentido la desesperación y frustración de encontrarnos en un atasco en algún momento. Pero los atascos tienen un gran impacto en el mundo además de agravar nuestra calidad de vida: cuestan dinero, mucho dinero.

En el estudio *The future economic and environmental costs of gridlock in 2030* [1] realizado por el CEBR (Centre for Economics and Business Research), se investigan los costes monetarios que tienen los atascos, así como el impacto medioambiental de los mismos. El estudio se centra en grandes países que sufren de este problema, como son Francia, Alemania, Gran Bretaña y los Estados Unidos, y tiene en cuenta tanto costes directos (gasto de gasolina y pérdida de tiempo) como indirectos (aumento de costes en muchos negocios).

El CEBR estimó que, en 2013, entre los cuatro países mencionados, el gasto total fue de 200 mil millones de dólares, y que en 2030 este gasto aumentará en un 46%, debido en gran parte al aumento de la población durante estos años. Además, estudió el impacto ambiental que suponen los atascos, al consumir innecesariamente gasolina y aumentar consecuentemente la contaminación por el CO_2 emitido. En 2013, se estableció que el total de litros consumido por vehículos durante los atascos en los países anteriormente mencionados fue de 5.950 millones de litros. Esto supone una emisión de alrededor de 15.400 kilotones de CO_2, una cantidad que aumentará también un 16% en los próximos años.

Así pues, uno puede hacerse una idea de la importancia que tiene disminuir en la medida de lo posible las congestiones de tráfico. Muchas de estas congestiones se producen en los núcleos urbanos de grandes ciudades, donde los semáforos cobran una gran importancia a la hora de gestionar el tráfico.

Sin embargo, la política fija de los semáforos puede ser mejorada por sistemas inteligentes como el que se va a desarrollar con este trabajo. Si se consigue mejorar la fluidez del tráfico con una mejor gestión de los semáforos, estaremos reduciendo los atascos y, por tanto, ahorrando una gran cantidad de dinero en combustible a la vez que se reducirá el impacto medioambiental.

El **objetivo principal** de este proyecto es desarrollar un sistema multi-agente distribuido en el que se utiliza aprendizaje automático por refuerzo para que cada semáforo sea capaz de actuar de manera independiente, cada uno de los cuales dispondrán solamente de información relativa a la vía que controla, sin conocer absolutamente nada de lo que ocurre en otros semáforos del cruce donde se encuentra y tomando decisiones en función de esta información, con el fin de mejorar la fluidez de todo el tráfico.

Para lograr esto, se han marcado una serie de **objetivos secundarios**:

- Integrar aprendizaje por refuerzo con el simulador SUMO.
- Desarrollar un agente individual e independiente que gestione un sólo semáforo mediante aprendizaje por refuerzo. Para ello:

    - Se tiene que diseñar un espacio de estados cuyos atributos sean locales, es decir, que pertenezcan únicamente al entorno del semáforo.
    - Se tiene que diseñar una función de refuerzo que también sea local, de forma que el agente solo puede aprender a partir de sus acciones y de la información disponible en su entorno.
- Entrenar este agente para que aprenda un comportamiento óptimo de gestión de tráfico.
- Replicar el agente desarrollado en todos los semáforos de una zona.
- Evaluar la fluidez del tráfico con este sistema con respecto al utilizado en la actualidad de política fija y con un sistema de política aleatoria.
    - Para ello, se deben utilizar métricas para medir la eficacia del sistema desarrollado que sean compatibles también con un sistema de política fija y política aleatoria.

De esta forma, se pretende crear un sistema que sea replicable en la realidad mediante una serie de sensores que ofrezcan toda la información local necesaria y que sea altamente escalable al poder replicar el agente en cualquier semáforo, sin importar la estructura del cruce donde esté instalado. Esto es posible gracias al diseño del espacio de estados y la función de refuerzo mencionado anteriormente, ya que, al tratar solamente información del ámbito local del agente, se consigue que el agente sea fácilmente replicable en cualquier situación. Y aunque el sistema es distribuido y cada agente sólo dispone de información local, se espera que emerja un comportamiento global adecuado.

Al conseguir el último de los objetivos secundarios mencionados, se podrá medir la fluidez del tráfico de forma objetiva y comparar los resultados de ambos sistemas. Además, las pruebas que se realicen deben de cubrir todos los casos de tráfico posibles, desde situaciones donde el tráfico sea casi inexistente hasta situaciones de tráfico congestionado. Así y solo así, podremos comparar realmente los sistemas en cuestión y llegar a algunas conclusiones.

Toda la información sobre cómo se han enfrentado estos problemas y se ha intentado conseguir los objetivos planteados se encuentra en el apartado *Descripción del trabajo realizado*. Mientras que la información relativa al comportamiento del sistema desarrollado se encuentra en el apartado *Experimentación y resultados*.

3. Estado del arte

En este apartado se describe la técnica de aprendizaje automático utilizada para desarrollar el agente: aprendizaje automático por refuerzo, en concreto, el método Q-learning. A continuación, se exponen varios trabajos realizados con estas técnicas y otras en el ámbito del control inteligente de semáforos.

Por último, se describen las herramientas utilizadas durante el proyecto: el simulador de tráfico SUMO, la aplicación web Open Street Map y el editor de mapas JOSM. Se discuten sus ventajas e inconvenientes y se elige un lenguaje de programación con el que desarrollar el agente de control de tráfico.

3.1. Aprendizaje automático

El **aprendizaje automático** es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a los ordenadores "aprender". Estas técnicas tratan de generar comportamientos a partir de experiencia en forma de ejemplos y se centran en resolver problemas de clasificación, optimización y predicción. Tienen una gran cantidad de aplicaciones: detección de fraude con tarjetas bancarias, reconocimiento del lenguaje escrito y hablado, robótica y diagnósticos médicos, entre otros muchos.

El aprendizaje automático se puede dividir en tres tipos:

- Aprendizaje supervisado: Técnica de aprendizaje automático cuyos ejemplos de entrenamiento se encuentran clasificados o etiquetados. Trata de generalizar a partir de estos datos a situaciones no presentadas anteriormente. El objetivo es etiquetar correctamente experiencias nuevas a partir de las dadas.
- Aprendizaje no supervisado: Técnica de aprendizaje automático cuyos ejemplos de entrenamiento no se encuentran clasificados. No existe un conocimiento a priori. Trata de agrupar los datos a partir de los atributos de los ejemplos.
- Aprendizaje por refuerzo: Técnica de aprendizaje automático cuyo objetivo es determinar las mejores acciones que puede tomar un agente en un entorno concreto para maximizar una recompensa.

En este proyecto se va a utilizar el aprendizaje por refuerzo, el cual se describe con mayor detalle en el siguiente apartado.

3.1.1. Aprendizaje por refuerzo

El **aprendizaje por refuerzo** se basa en la idea de un agente que debe aprender un comportamiento por ensayo y error interactuando con un entorno. El modelo del aprendizaje por refuerzo consta de varios elementos:

- Un conjunto de estados que representa el entorno.
- Un conjunto de acciones que pueda realizar el agente.

- Una función de refuerzo que defina la recompensa recibida por el agente al realizar una acción sobre el entorno.

La interacción entre el agente y el entorno está representada en la Ilustración 5:

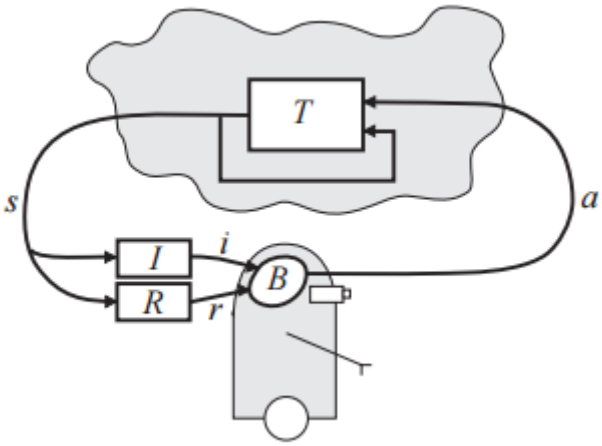

Ilustración 5: Modelo estándar de aprendizaje por refuerzo. Extraído de [2]

El agente (representado con una 'B' en la imagen) toma una acción 'a' que se ejecuta en el medio 'T'. De este medio proceden los estados que contienen información sobre el mismo y la recompensa de la acción llevada a cabo. Este procedimiento es comúnmente llamado ciclo.

El agente utiliza la información recogida durante un conjunto de ciclos para aprender un comportamiento, determinando en qué situaciones las acciones llevadas a cabo anteriormente son beneficiosas o no.

Cada uno de los ciclos viene representado por tuplas. Una tupla es el conjunto de un estado en un tiempo t, la acción llevada a cabo, el estado en el tiempo t+1 y el refuerzo del comportamiento elegido. Este ciclo se vuelve a repetir N veces, de forma que el agente explore distintos comportamientos hasta descubrir uno óptimo.

El refuerzo dependerá del comportamiento que queramos conseguir con el agente. Típicamente, este refuerzo está comprendido en un intervalo entre un valor negativo y un valor positivo, aproximándose al valor mínimo el refuerzo obtenido por las acciones que más alejen al agente del comportamiento elegido y al valor máximo las que se quiere conseguir aprender.

Adicionalmente se suelen aplicar políticas de exploración y explotación que afectan a la decisión del agente sobre qué acción tomar. Una de las políticas más utilizadas es la **ε-greedy**. Esta política define una probabilidad ε de elegir la mejor acción de entre todas las acciones, otorgando así una probabilidad (1 - ε) de elegir una acción aleatoria. Esta variable ε posee un valor bajo durante la etapa de exploración, con el objetivo de que el agente tome acciones

aleatorias y explore el espacio de estados. A medida que avanza el aprendizaje, este valor se incrementa y pasamos a la fase de explotación, donde el agente escoge las mejores acciones.

Esta política será la utilizada para este proyecto, ya que se han obtenido buenos resultados en otros estudios con esta misma estrategia.

Q-Learning

Q-learning es una técnica de aprendizaje por refuerzo introducida por Watkins en 1989. Esta técnica dota al agente de la capacidad de aprender a actuar de forma óptima procesos de decisión de Markov (MDPs) a través de la experiencia. El agente toma acciones y recibe una recompensa. De esta forma, no se requiere describir todo el dominio, ya que el agente lo explorará en busca de optimizar su comportamiento.

Sin embargo, sólo está comprobado que encuentre una solución óptima en el caso de espacio de estados y acciones finitos [3]. Esto es aplicable también en espacio de estados y acciones no finitos, los cuales los podemos representar de forma finita tras aplicar un proceso de discretización.

El algoritmo consta de varios elementos: un conjunto discreto de estados, un conjunto discreto de acciones, una función de refuerzo, una tabla donde se guardan los resultados del aprendizaje y una función que define cómo se actualiza esta tabla.

El conjunto de estados, acciones y la función de refuerzo vienen definidos por el dominio del problema. El diseño de estos elementos son la parte más importante para que el sistema funcione correctamente. En los apartados *Descripción del espacio de estados*, *Descripción del espacio de acciones* y *Descripción de la función de refuerzo*, vienen descritos en detalle estos elementos.

Como en todo aprendizaje por refuerzo, este método se basa también en las tuplas de entrenamiento, que contienen la siguiente información: estado actual, acción, estado siguiente y refuerzo. Mediante los valores que tomen los atributos de esta tupla, se actualiza la tabla que almacena los resultados (la tabla Q) mediante la ecuación 2:

Ecuación 2: Actualización de la tabla Q

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

Donde cada uno de los elementos que componen la ecuación se describen a continuación:

- $Q(s_t, a_t)$: Valor de la tabla Q para el estado s y la acción a en el tiempo t.

- α: Ratio de aprendizaje. Determina la variación en la tabla Q a partir de la nueva experiencia. Un valor de 0 hará que el agente no aprenda nada, mientras que un valor de 1 hará que el agente sólo considere la información más reciente.
- r: Refuerzo obtenido tras aplicar la acción a en el estado s en el tiempo t.
- γ: Factor de descuento. Determina la importancia de futuras recompensas. Un factor de 0 hará que el agente sólo considere recompensas recientes, mientras que un factor de 1 hará que el agente busque recompensas a largo plazo.
- $\max_a \mathrm{Q}(s_{t+1}, a)$: Mayor valor de la tabla Q para el estado s.

Como se puede observar, la tabla Q se modifica dependiendo de la experiencia previa, el refuerzo obtenido en un instante y una estimación del refuerzo que se obtendrá en un futuro, utilizando para controlar los cambios dos parámetros: alpha (ratio de aprendizaje) y gamma (factor de descuento). Estos parámetros tienen un gran impacto en el comportamiento del agente. El proceso de obtener los mejores parámetros para resolver este problema se encuentra descrito en la sección *Descripción de los parámetros del Q-learning*.

La tabla Q obtenida dispondrá de tantas filas y columnas como estados y acciones se hayan definido. A partir de esta tabla se decidirá qué acción tomar en cada situación. Durante el aprendizaje, se tendrá que definir una política que determine cómo se eligen estas acciones. Por ejemplo, estas acciones se pueden elegir de forma aleatoria para explorar el espacio de estados o se puede elegir siempre la mejor acción (aquella que tenga un mayor valor para un estado s) para evaluar el comportamiento aprendido. Como ya se ha mencionado anteriormente, en nuestro caso escogeremos una política ε-greedy.

3.1.2. Otras técnicas de aprendizaje automático

Existen muchas otras técnicas de aprendizaje automático que pueden ser aplicadas a este mismo problema además de la técnica elegida. A continuación, se presentan algunas de éstas:

Se pueden utilizar **árboles de decisión** como modelo de comportamiento de los semáforos. Un ejemplo de este tipo de algoritmos es el ID3. Como ventaja presenta la claridad del modelo, pero tiene una clara desventaja: los datos de ejemplo hay que etiquetarlos. Al abordar el problema del tráfico, se parte de la premisa de no saber cuál es la acción óptima para manejar un semáforo, por lo que etiquetar correctamente los datos sería una tarea difícil. Otra manera de intentar abordar el problema es generando directamente árboles de decisión y probarlos en simuladores. Un ejemplo de un estudio que ha abordado el problema del tráfico con árboles de decisión es el *ETC assisted traffic light control scheme [4]*.

En una situación similar a la anterior se encuentran las **redes de neuronas**. Aunque la claridad del modelo aquí es nula, lo cual es una desventaja, esta técnica presenta la ventaja de ser más robusta a ruido en los datos de aprendizaje. Las redes de neuronas podrían ser utilizadas de dos formas distintas: en una se intentaría clasificar si para una situación dada es mejor abrir o cerrar un semáforo, y en otra se intentaría predecir la fluidez del tráfico al realizar ambas

opciones y escoger la mejor. Un ejemplo de un estudio donde se aplicó esta técnica en el control de semáforos se puede encontrar en el artículo *Neural Networks for Real-Time Traffic Signal Control* [5].

Otra técnica de aprendizaje automático utilizada en este ámbito son los **algoritmos genéticos**. Éstos se aplican en problemas similares al que queremos resolver con un gran éxito: aprendizaje de comportamiento de robots, problemas de optimización y distribución de energía eléctrica. Así como con las otras técnicas, también existen estudios donde se han aplicado este tipo de algoritmos en el problema del control de semáforos. Un ejemplo es el artículo *Traffic Control with Standard Genetic Algorithm* [6].

3.2. Control automático de semáforos

En la actualidad el uso de sistemas inteligentes de control de semáforos no está muy extendido. Aunque existen muchos sistemas, son muy caros de instalar y su rentabilidad es dudosa.

Sin embargo, en algunos países como Estados Unidos, existen varios sistemas de control de tráfico funcionando por gran parte de los estados que lo componen, aunque suelen estar ubicados en escenarios específicos y no están muy extendidos.

En el estudio *Adaptive Traffic Control Systems in the United States* [7] se hace una comparación entre estos sistemas: InSync, ACS-Lite y LA ATCS. En el documento se incluye el coste de despliegue del sistema por intersección, siendo InSync el más barato con un coste medio de 28.700$ por intersección. También se compara el mantenimiento necesario de cada sistema. InSync requiere menos tiempo de mantenimiento que los sistemas comparados, con tan solo 18 minutos de mantenimiento a la semana por cada intersección.

La ilustración 6 muestra la mejora del tráfico tras instalar estos sistemas, siendo InSync otra vez el que obtiene mejores resultados.

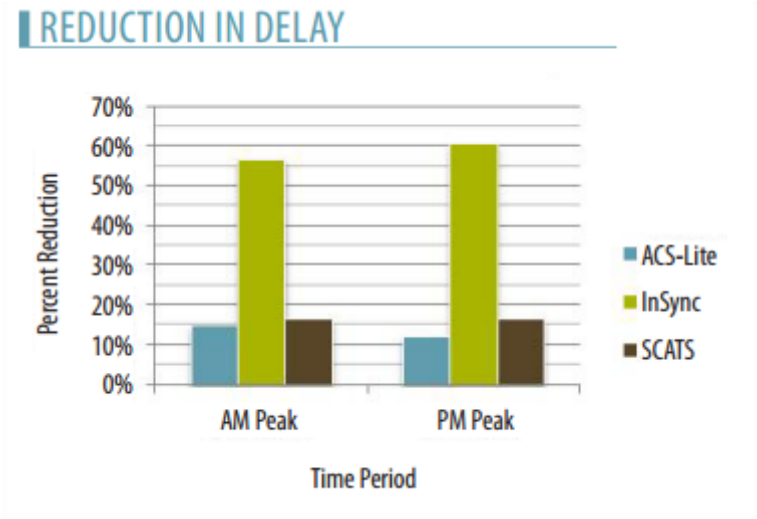

Ilustración 6: Comparativa de sistemas inteligentes de tráfico. Extraído de [7]

Otro ejemplo de sistema inteligente de control de tráfico es SURTRAC (Scalable Urban Traffic Control), desplegado en Pittsburgh, Pennsylvania. SURTRAC se basa en la planificación automática para gestionar los semáforos. Este sistema tiene algunas ventajas: es capaz de adaptarse en tiempo real al tráfico de la vía y es un sistema descentralizado, lo cual le dota de una gran escalabilidad. Además, como se puede ver en la ilustración 7, el sistema mejoró sustancialmente la fluidez del tráfico:

TABLE 1 Summary of pilot test results

Percent improvement	Average Vehicles	Travel time	Speed	Number of Stops	Wait Time	Emissions
AM rush	5,228	30.11%	33.78%	29.14%	47.78%	23.83%
Mid Day	8,007	32.83%	48.55%	52.58%	49.82%	29.00%
PM rush	9,548	22.65%	27.45%	8.89%	35.60%	18.41%
Evening	7,157	17.52%	27.81%	34.97%	27.56%	14.01%
Overall	29,940	25.79%	34.02%	31.34%	40.64%	21.48%

Ilustración 7: Resultados del sistema SURTRAC. Extraído de [8]

A pesar de esto, SURTRAC recibió muchas críticas: utiliza cámaras para obtener información del cruce, lo cual daña gravemente la privacidad, y el sistema requería que los peatones pulsaran siempre un botón para poder cruzar, quedando así relegados a un segundo plano, aumentando la espera que sufren los transeúntes.

Con estos ejemplos se puede ver que los sistemas inteligentes de control de tráfico no son perfectos. Tienen un coste relativamente alto y aunque los resultados son buenos, éstos siempre se pueden mejorar. Por ello, siguen publicándose estudios en esta área. En [9] y [10], se pueden consultar estudios sobre sistemas que aplican aprendizaje por refuerzo en este mismo problema.

3.3. SUMO

SUMO (Simulation of Urban MObility) [11] es un simulador de tráfico gratuito y abierto que permite modelar sistemas de tráfico intermodales, incluyendo carreteras, transporte público e incluso peatones. Dispone de herramientas para visualizar la simulación, para importar una red de carreteras y para calcular las emisiones de los vehículos. Además, existen interfaces que te permiten controlar la simulación, como TraCI.

Una de las características más importantes de SUMO es que te permite cambiar las luces de los semáforos a tu voluntad y no existe un límite de coches simulados, lo cual encaja perfectamente con el propósito de este proyecto.

La interfaz TraCI te permite además consultar el estado de un semáforo, cambiarlo, consultar el número de vehículos en una vía, la velocidad media y máxima en la misma, y el tiempo de espera de los vehículos en un semáforo, entre otras muchas funcionalidades. Esta interfaz soporta varios lenguajes de programación: Python, Java, Matlab y C++.

El simulador dispone de una GUI en la que se muestra la simulación, como se puede ver en la ilustración 8:

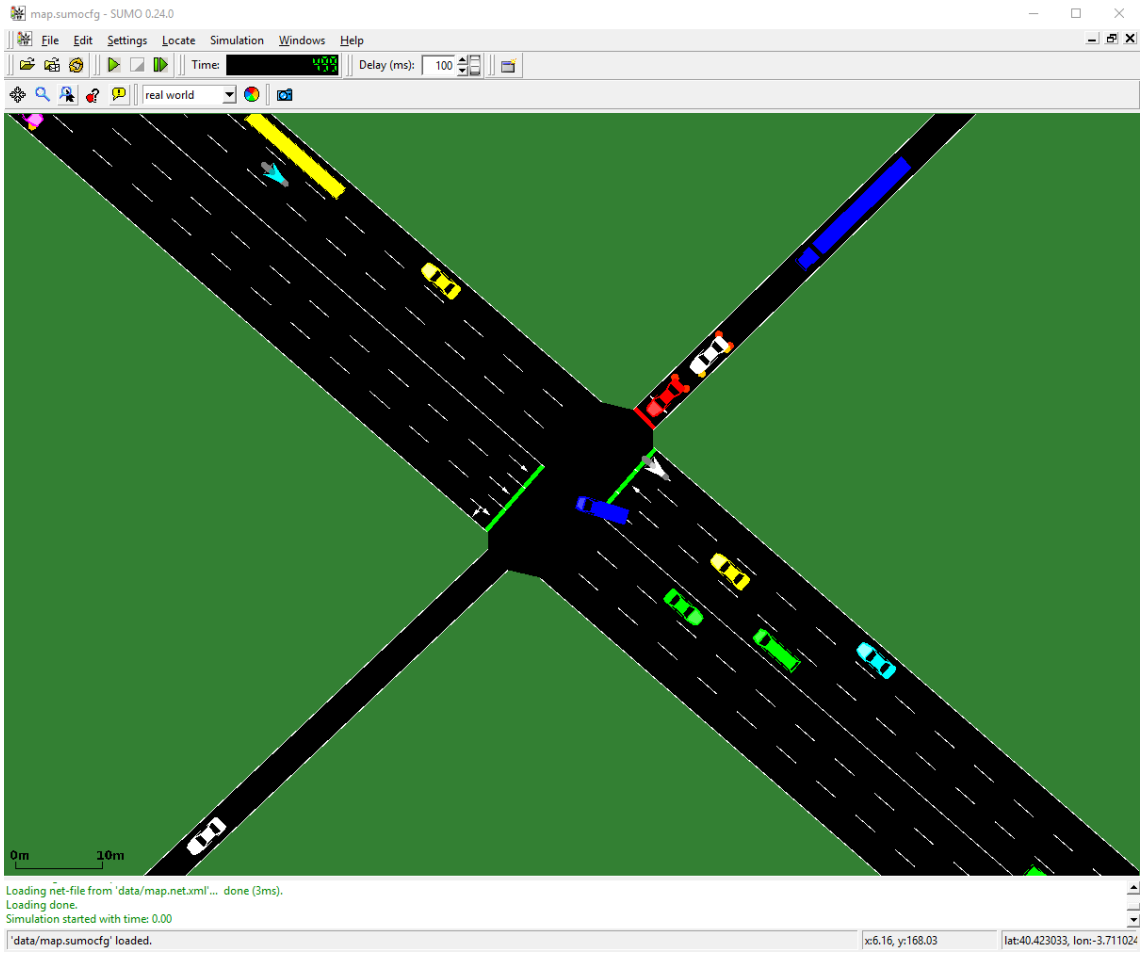

Ilustración 8: Interfaz de SUMO

En la parte superior de la interfaz, debajo de los menús de la aplicación, se encuentran: una serie de botones que sirven para abrir simulaciones desde fichero y recargar una simulación; otros botones para controlar la simulación (empezar la simulación, pararla e ir paso a paso); un contador que muestra la cantidad de pasos de simulación que se han llevado a cabo hasta ese momento; y un indicador del retraso en milisegundos entre cada paso de la simulación, el cual es modificable. Además de todos estos botones, en la parte inferior de la interfaz se muestra una consola. Aquí se podrán ver todo tipo de mensajes relevantes para la simulación: errores, notificaciones cuando un vehículo lleve esperando demasiado tiempo y cualquier otro tipo de alertas.

En la parte central de la interfaz se puede observar la simulación. En esta interfaz se distinguen varios elementos: la red de carreteras, las marcas viales, los diferentes tipos de vehículos (coches, motos, camiones…) y los semáforos. La apariencia de todos estos elementos se puede modificar mediante el botón situado debajo del contador del tiempo.

Para que el simulador funcione, se requieren varios ficheros: un fichero .net que contenga toda la información del mapa donde se va a hacer la simulación (las vías, especificando para cada una el tipo de vía que es y la velocidad máxima de la misma, los cruces, los giros permitidos y la lógica de los semáforos); un fichero .rou que contenga todas las rutas de cada uno de los

vehículos que van a participar en la simulación, incluyendo el tipo de vehículo que es cada uno, su color y momento en el que debe iniciar su viaje; y un fichero .settings que contiene diferentes configuraciones para la GUI (posición en el mapa de la cámara, el zoom y el 'delay' por defecto).

El simulador, por su parte, genera un fichero llamado tripinfo.xml con toda la información de cada uno de los viajes realizados por cada vehículo: en qué momento inicia su viaje, dónde, cuál es su destino, cuándo llega al destino, la longitud de la ruta y el tiempo perdido esperando.

SUMO es una herramienta adecuada para trabajar con simulaciones de tráfico. Tiene un motor que ofrece simulaciones de calidad, una interfaz sencilla y es altamente configurable, ya que puede funcionar con cualquier red de carreteras que exista y se puede cambiar hasta la aceleración, deceleración y tamaño de todos los vehículos.

En contrapartida, tiene algunas desventajas. Es imposible retroceder en una simulación y exige una gran cantidad de trabajo para preparar los mapas y el tráfico que utiliza. A pesar de esto, es uno de los mejores simuladores de tráfico existentes y ha sido utilizado en muchos proyectos, como se puede consultar en su página web.

3.4. Python

Python es uno de los lenguajes de programación soportados por la interfaz TraCI de SUMO. Es un lenguaje de alto nivel que destaca sobre sus competidores por la claridad y simplicidad del código generado.

Además de su sencillez, muchos programas escritos en Python tienen un menor número de líneas que el mismo programa escrito en lenguajes como Java o C++. Python destaca también por consumir mucha menos memoria que Java.

A todas estas ventajas se suma que la mayor parte de la documentación de SUMO se encuentra escrita en Python.

Por todas estas ventajas, se ha elegido Python como lenguaje de programación para desarrollar este proyecto.

3.5. Open Street Map

El simulador necesita un mapa con el que trabajar y en el que simular el tráfico. Aunque se pueden generar estos mapas con algunas herramientas, siempre es mejor trabajar con mapas reales, ya que son con los que obtendremos resultados más fiables.

Open Street Map [12] es una aplicación web gratuita que te permite descargarte mapas de ciudades reales. En la ilustración 9, se muestra la interfaz de la aplicación, donde se puede ver que contiene información de una gran cantidad de ciudades:

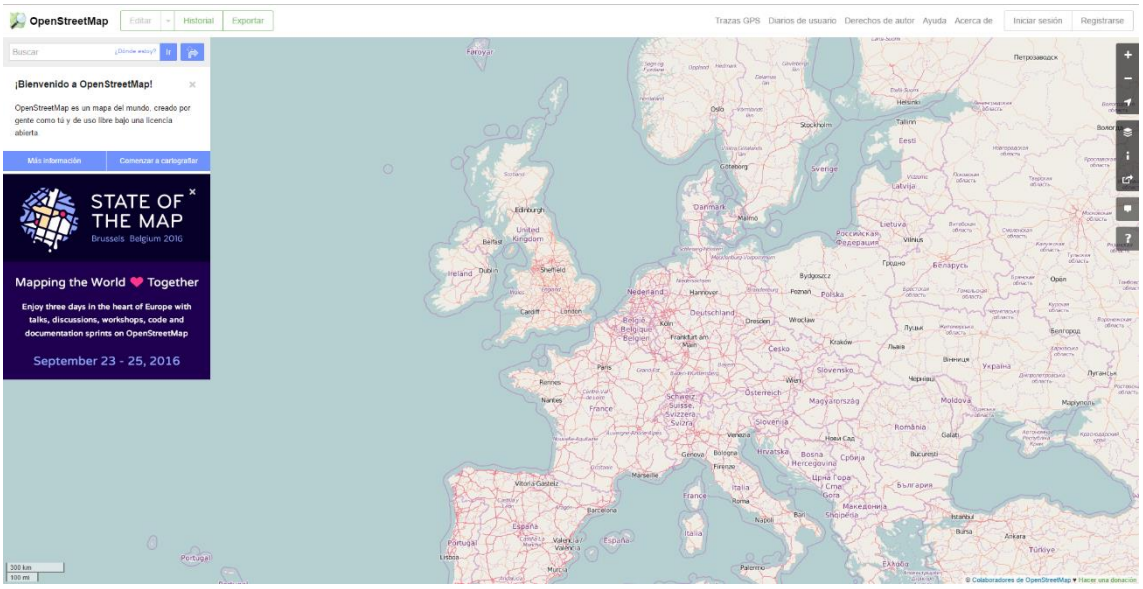

Ilustración 9: Interfaz de OpenStreetMap

Esta interfaz dispone de un buscador en la parte izquierda que te facilitará la tarea de encontrar el lugar que deseas. En este caso, vamos a buscar una de las vías más importantes de España, la Gran Vía.

Una vez encontrado el lugar, puedes delimitar manualmente la zona que deseas descargar accediendo al menú “Exportar” de la parte superior izquierda de la pantalla, como se puede observar en la ilustración 10. Realizando esta acción, se generará un archivo con formato osm que contendrá toda la información disponible en la aplicación: nombre de las vías, paradas de transporte público, pasos de cebra, semáforos y distintos puntos de interés.

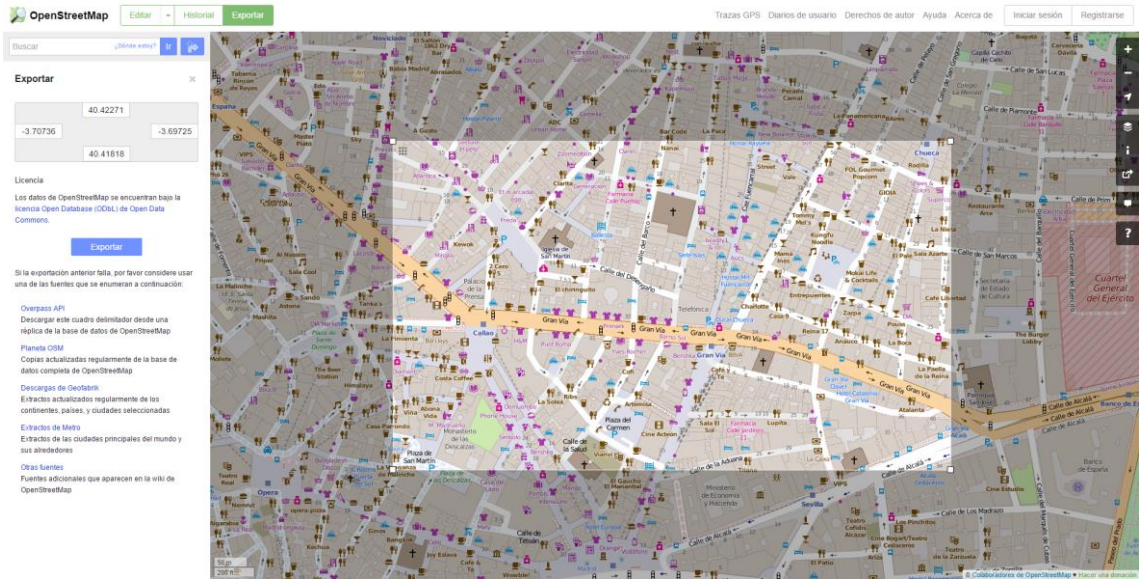

Ilustración 10: Exportar mapa en OpenStreetMap

Este programa facilita en gran medida la generación de sistemas de tráfico realistas, al poder descargar directamente una red de carreteras real. Sin embargo, presenta demasiada información que es irrelevante para el caso en el que estamos interesados y se necesita editar el mapa 3D generado para que sea apto para funcionar con el simulador SUMO.

3.6. JOSM

JOSM [13] es un editor Java para Open Street Map. Gracias a este editor podremos modificar el archivo que hemos descargado desde la aplicación web de Open Street Map. JOSM ofrece una gran cantidad de posibilidades para modificar los mapas, pero aquí sólo se detallarán las partes más importantes de la interfaz del programa.

Como se puede ver en la ilustración 11, en la parte superior se encuentran los botones para abrir el fichero osm que contiene toda la información descargada de la aplicación web. En la parte izquierda se encuentran distintas herramientas de edición para crear y modificar carreteras, así como para abrir y cerrar los menús que se encuentran visibles en la parte derecha de la interfaz. Estos menús muestran información como la relación entre las calles, los atributos de una vía seleccionada o los errores, si existen, entre otros.

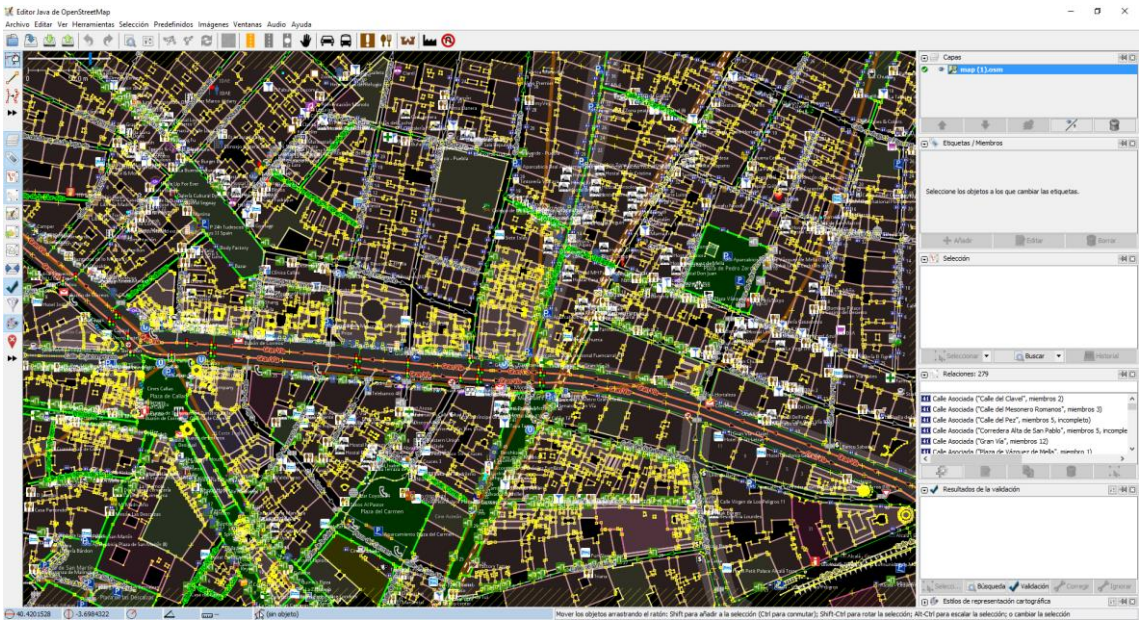

Ilustración 11: Interfaz de JOSM

El fichero descargado contiene información muy detallada que no nos aporta nada en la simulación de tráfico que queremos realizar. Así, la localización de las tiendas y los puntos de interés, las paradas de bus, metro y tren, y todos los edificios pueden ser eliminados. El fichero contendrá solamente información relativa a las carreteras descargadas.

Una vez hecho este proceso, se tendrán que arreglar los errores que surjan. Estos errores suelen estar relacionados con vías no conectadas con ninguna otra, nodos duplicados, vías sin etiquetar, vías sin nombre y errores en la relación entre las vías, entre otros problemas.

Todo este procedimiento es necesario para poder trabajar en SUMO con un mapa real. Sin embargo, todo este trabajo es recompensado al poder realizar simulaciones en situaciones reales, obteniendo así resultados más prácticos que si se hubieran utilizado mapas sintéticos.

En resumen, como se ha demostrado en este apartado sobre el Estado del arte, existen una variedad de sistemas similares al que aquí se plantea que se encuentran en funcionamiento en ciertas localidades. Sin embargo, ninguno es perfecto: algunos no son escalables y otros son muy caros. También se ha visto que se dispone de todas las herramientas necesarias para desarrollar este proyecto, por lo que se concluye que el proyecto es viable y que servirá para investigar la utilidad de un sistema multi-agente distribuido para el control del tráfico.

4. Gestión del proyecto

En este apartado se encuentra toda la información relacionada con la gestión del proyecto. Para llevar a cabo un proyecto software con éxito, es necesario definir la metodología a seguir, así como las etapas en las que se divide su desarrollo, realizando una planificación de cada una de éstas con el objetivo de conseguir los objetivos en un tiempo establecido. Además, se incluye el presupuesto del proyecto, el cual detalla los costes en personal y recursos previstos para este trabajo.

4.1. Metodología

Todo proyecto software debe seguir una metodología para estructurar y planificar el desarrollo del sistema. Esta metodología divide el desarrollo en varias etapas. Así, es más sencillo alcanzar los objetivos del proyecto.

Existen muchas metodologías de desarrollo software. Por ejemplo, el **prototipado**, con el cual se obtiene una versión del software con las funciones más básicas y a través de varias iteraciones con clientes, se consigue incrementar la funcionalidad del mismo, cumpliendo con las expectativas de los mismos.

Otro ejemplo es **SCRUM**, que es una metodología ágil de desarrollo cada vez más utilizadas en muchas empresas. SCRUM se parece al prototipado en el sentido de que también trata de ser un proceso iterativo. Sin embargo, este modelo es mucho más flexible en cuanto a planificación, centrándose en dividir el trabajo en tareas muy pequeñas y realizarlas en el menor tiempo posible, respondiendo así a los requerimientos del cliente de forma rápida. El trabajo en equipo también cobra mucha importancia, siendo esencial realizar reuniones de todo el equipo con asiduidad.

El **modelo en cascada** es otra metodología muy extendida, llamada así porque las fases parecen caer por gravedad una sobre otra. Este modelo es mucho más rígido que los anteriores, define con mayor claridad cada etapa del proceso y cada una de éstas empieza al finalizar la anterior.

Las etapas en las que este modelo divide el desarrollo, como se puede ver en la ilustración 12, son: Análisis de requisitos, diseño del sistema, implementación del mismo, pruebas y, por último, mantenimiento.

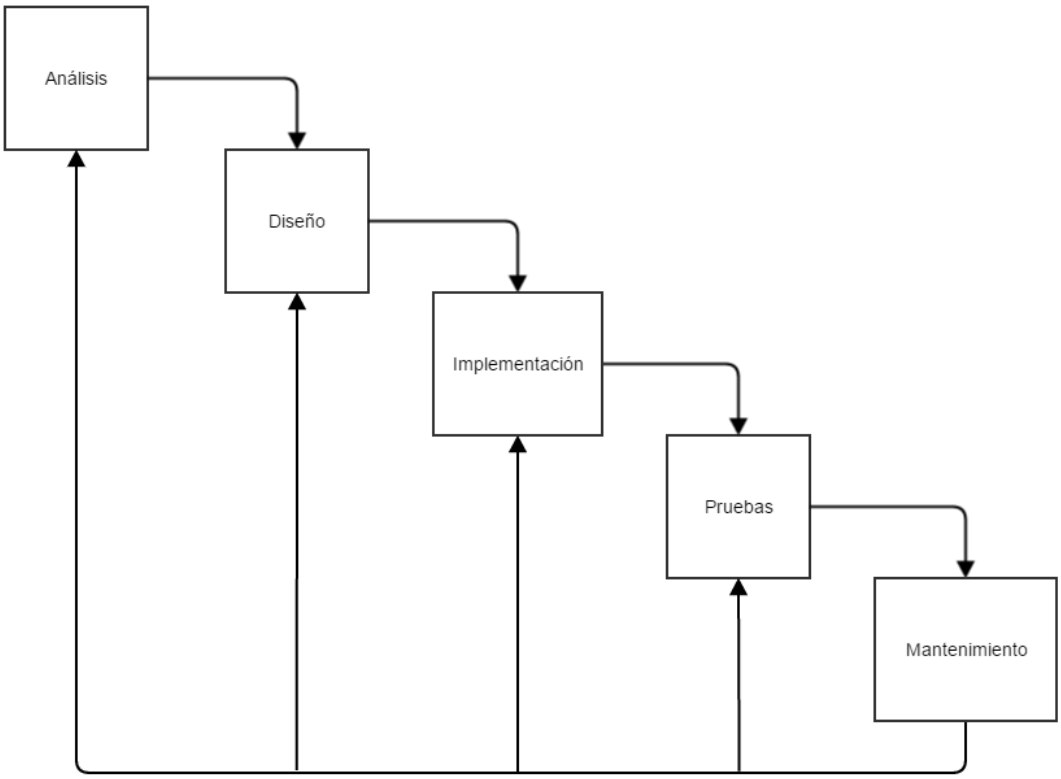

Ilustración 12: Modelo en cascada

A continuación, se van a definir con más detalle cada una de estas etapas:

Análisis de requisitos

En esta etapa se analizan las necesidades a cubrir y se documentan los requisitos del sistema en una memoria que contiene la especificación de lo que debe hacer el mismo. Los requisitos especificados para este proyecto se pueden consultar en el siguiente apartado.

Es importante realizar un buen análisis de los requisitos desde un principio, ya que si hace falta añadir algún requisito cuando el proyecto se encuentra en cualquiera de las otras etapas, supondrá volver a pasar otra vez por todas las etapas.

Diseño del Sistema

Este proceso establece la arquitectura del sistema, descomponiéndolo en distintos elementos. De esta forma, se aumenta la reusabilidad y mantenibilidad de los mismos, a la vez que se impulsa el trabajo en equipo al poder separar el trabajo de forma clara. Durante esta etapa, se realiza la especificación de cada uno de los componentes del sistema y de la relación entre los mismos.

El diseño del software de este proyecto se puede consultar en el apartado llamado *Diseño del software*.

Implementación

En esta etapa se implementa el sistema diseñado en un lenguaje de programación (o varios) de forma que se cumplan con los requisitos anteriormente especificados. Durante esta fase se utilizan técnicas de programación para producir componentes reutilizables de forma que este proceso sea mucho más rápido en un futuro.

Pruebas

Esta es una de las etapas más importantes. En ésta se comprueba que el sistema funcione correctamente y que cumpla con todos los requisitos especificados.

Gracias a esta etapa se pueden logran productos de calidad. Disponer de una gran variedad de pruebas que comprueben el correcto funcionamiento del sistema en profundidad, puede ahorrar tiempo y costes, a la vez que evita la pérdida de prestigio en caso de que se despliegue un software con errores.

Dentro de esta fase suele incluirse la verificación del producto. Aquí es el usuario final el que comprueba que el sistema funciona correctamente y que cumpla con sus expectativas.

Mantenimiento

Se podría decir que es la etapa más crítica por la naturaleza del modelo en cascada, al ser la última etapa. Una vez desplegado el sistema, se debe arreglar cualquier error que se detecte. En el caso de que existiera algún error, supondría volver a pasar otra vez por varias de las etapas anteriores (dependiendo del error, serían más o menos etapas), aumentando así los costes de desarrollo del producto. Por esta razón, es importante realizar correctamente todas las anteriores etapas.

Este modelo presenta algunas desventajas. La más importante es que si tienes que volver a realizar alguna etapa, tendrás que volver a realizar todas las siguientes. Así, cualquier pequeño cambio puede generar una gran cantidad de trabajo.

Sin embargo, a pesar de esta desventaja, se ha elegido esta metodología por estar familiarizado con ella, por su sencillez y por conocer con cierta exactitud el producto final (no se van a realizar cambios en el software en una etapa avanzada del desarrollo del mismo).

4.2. Planificación

En este apartado se va a describir la **planificación** del proyecto. Ésta divide la completitud del trabajo en distintas fases, determinando para cada una de ellas una fecha de principio y fin. El objetivo de esta tarea es completar el trabajo en un tiempo determinado, otorgando a cada actividad del tiempo justo y necesario para que sea acabada.

La metodología a seguir, que en este caso es el modelo en cascada, se puede ver reflejada claramente en el plan de trabajo. Cada actividad sucede a la anterior y se basa en el resultado de ésta para seguir realizando el proyecto de forma incremental.

Las actividades en las que se ha dividido el proyecto son las siguientes:

- El **estudio del problema**. Esta es la fase donde se lleva a cabo el estudio del arte. Se buscan trabajos similares, técnicas utilizadas en este ámbito, se cuestiona la viabilidad del sistema y se buscan programas útiles para desarrollar el proyecto.

- La **especificación de requisitos**. Una vez estudiado el problema, éste se traduce en requisitos software que conforman el pilar sobre el que debe construirse el sistema. En esta fase se precisan estos requerimientos de la forma más clara y concisa posible.

- El **diseño del sistema**. Basándose en los requisitos especificados, y una vez se sabe qué debe hacer el sistema y cómo, se realiza un diseño del mismo que permita conseguir los objetivos. Así, en esta actividad se intenta dar con una solución que facilite la siguiente etapa: la implementación.

- La **implementación** del sistema. Durante esta tarea se programan todos los módulos del sistema indicados en la etapa anterior.

- Las **pruebas unitarias**. En esta actividad se realizan las pruebas necesarias para comprobar que cada uno de los módulos del sistema funcionan correctamente de forma individual.

- Las **pruebas de integración**. En esta actividad se realizan las pruebas necesarias para comprobar que todos los módulos del sistema funcionan correctamente en conjunto.

- La **planificación de la experimentación**. Una vez se ha asegurado el correcto funcionamiento del programa implementado, se llega a la fase de organizar los distintos escenarios donde se debe probar el sistema. Estos experimentos deben cubrir las distintas situaciones que el sistema debería enfrentar en caso de desplegarse y así obtener los resultados del comportamiento del mismo en cada uno de estos sucesos.

- La **experimentación**. En esta fase se somete el sistema a las situaciones descritas en la anterior actividad con el fin de comprobar si se consiguieron los objetivos planteados antes de desarrollar el mismo.

- La preparación de la **memoria**. Esta actividad es paralela a casi todas las anteriores. Su objetivo es plasmar en el documento que estás leyendo todo el trabajo realizado, en la medida de lo posible.

En la ilustración 13 se refleja la distribución del tiempo que se dedicará para cada una de estas actividades por semanas.

Actividad	Mes 1				Mes 2			
	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8
Estudio del problema								
Especificación de requisitos								
Diseño del sistema								
Implementación								
Pruebas unitarias								
Pruebas de integración								
Planificación de experimentación								
Experimentación								
Preparación de memoria								

Actividad	Mes 3				Mes 4			
	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
Estudio del problema								
Especificación de requisitos								
Diseño del sistema								
Implementación								
Pruebas unitarias								
Pruebas de integración								
Planificación de experimentación								
Experimentación								
Preparación de memoria								

Ilustración 13: Planificación del proyecto

De esta forma, se intentará realizar el proyecto entero en el tiempo descrito, siguiendo la metodología elegida y respetando cada una de las etapas de desarrollo software definidas anteriormente.

4.3. Presupuesto

Una vez se conoce la planificación del proyecto, se pueden estimar las horas de trabajo necesarias para desarrollarlo. Gracias a esto, se puede confeccionar un presupuesto ajustado al proyecto.

A continuación, se presentan los costes del personal, los bienes tangibles y los costes indirectos, seguidos de un pequeño resumen de los costes donde se expone el precio final del proyecto.

4.3.1. Coste del personal

Este trabajo requiere de cinco distintos profesionales para ser desarrollado con éxito: un director de proyecto, que se encargará de dirigir al equipo y tratará de cumplir los plazos establecidos; un analista, encargado de obtener los requisitos software; un diseñador, que elaborará el diseño del sistema; un programador, que implementará el programa; y un asegurador de calidad software, encargado de realizar las pruebas necesarias para comprobar que el sistema desarrollado funciona correctamente y cumple las expectativas.

A continuación, en la tabla 1 se resume el coste de todos estos trabajadores, desglosando para cada uno de ellos el coste por hora y las horas que se estima que trabajarán.

Número de trabajadores	Puesto de trabajo	Coste por hora	Horas	Total
1	Director del proyecto	36,00 €	47	1.692,00 €
1	Analista	24,00 €	56	1.344,00 €
1	Diseñador	23,00 €	43	989,00 €
1	Programador	15,00 €	112	1.680,00 €
1	Asegurador de calidad software	27,00 €	65	1.755,00 €
Subtotal				7.460,00 €
Total (incluyendo Seguridad Social)				**9.698,00 €**

Tabla 1: Coste del personal

El coste total del personal asciende a 9.698,00 €, precio que incluye la Seguridad Social de cada empleado, lo cual representa un 30% de los sueldos.

4.3.2. Coste de los bienes tangibles

Además del coste del personal, se ha de tener en cuenta el coste de los materiales con los que se va a trabajar. En este caso, cada empleado necesitará un portátil equipado con Windows 10 y el paquete Office de Microsoft. También se incluye en el presupuesto una impresora, para imprimir toda la documentación necesaria del proyecto.

En la tabla 2 se detallan todos los costes materiales, desglosados por precio y cantidad necesaria de cada uno de éstos.

Artículo	Precio por artículo	Cantidad	Total
Portátil Acer Aspire ES1-571-507Y	449,00 €	5	2.245,00 €
Microsoft Windows 10	135,00 €	5	675,00 €
Microsoft Office 2016	99,00 €	5	495,00 €
Impresora Multifunción HP Envy 5544	62,99 €	1	62,99 €
		Total	**3.477,99 €**

Tabla 2: Coste de los materiales

Sin embargo, el coste de estos materiales es menor debido al proceso de amortización que sufren. Este proceso hace referencia a la depreciación de los artículos con el paso del tiempo, y tiene en cuenta la vida de los materiales y su coste inicial.

En el ámbito del software, los equipos y programas informáticos tienen un periodo de amortización mínimo de 3 años y disponen de un máximo del 33% de amortización sobre el precio inicial.

A continuación, en la tabla 3, se especifica la amortización por cada artículo, según lo explicado anteriormente.

Artículo	Amortización por artículo	Cantidad	Periodo de amortización	Total
Portátil Acer Aspire ES1-571-507Y	148,17 €	5	3 años	740,85 €
Microsoft Windows 10	44,55 €	5	3 años	222,75 €
Microsoft Office 2016	32,67 €	5	3 años	163,35 €
Impresora Multifunción HP Envy 5544	20,79 €	1	3 años	20,79 €

Total	**1.147,74 €**

Tabla 3: Amortización de los materiales

De esta forma, se espera una amortización de 1.147,74€ de los materiales, lo cual será descontado del presupuesto final del proyecto.

Además de los materiales, el personal necesitará de un lugar donde trabajar. Por esta razón, se han añadido al presupuesto los costes de alquilar un inmueble. La tabla 4 refleja estos gastos, los cuales incluyen el alquiler, la luz, el agua, el gas e Internet.

Alquiler de inmueble			
Coste mensual	**Gastos mensuales**	**Meses**	**Total**
580,00 €	190,00 €	4	**3.080,00 €**

Tabla 4: Alquiler del inmueble

Así, se espera gastar un total de 3.080,00 € en el inmueble durante la duración del desarrollo del sistema.

4.3.3. Costes indirectos

Los costes indirectos son aquellos que pueden surgir durante el desarrollo del producto pero que no pueden ser asignados directamente al mismo. Esto engloba daños materiales, subidas en impuestos y gastos administrativos y financieros de la empresa. Para este proyecto, se ha establecido que los costes indirectos sean del 10% del coste del mismo para poder cubrir todos estos pagos.

4.3.4. Precio total

A continuación, se presenta el resumen de los costes del proyecto en la tabla 5. Éstos incluyen los costes de personal, de los materiales y el inmueble. Además, se ha descontado la amortización de los materiales y se han añadido los costes indirectos y el Impuesto sobre el Valor Añadido (I.V.A.).

Coste total del proyecto	
Concepto	**Coste**

Personal	9.698,00 €
Materiales	3.477,99 €
Amortización de materiales	- 1.147,74 €
Inmueble	3.080 €
Subtotal	15.108,25 €
Costes indirectos (10%)	1.510,83 €
I.V.A (21%)	3.172,74 €
Total	**19.791,82 €**

Tabla 5: Coste total del proyecto

El **precio final** del proyecto, como se puede ver en la tabla superior, es de **19.791,82 €**, con I.V.A. incluido. Este presupuesto es válido durante tres meses, contando desde el día de su presentación al cliente. Una vez pasado este tiempo, este presupuesto podría cambiar.

5. Descripción del trabajo realizado

En este apartado se va a detallar todo el trabajo realizado durante la realización de este proyecto, desde la especificación de requisitos y el diseño del software implementado hasta el diseño del aprendizaje automático y la fase de experimentación.

5.1. Requisitos software

En este apartado se van a definir los requisitos software de este proyecto. Estos requisitos hacen referencia a distintas características de sistema: qué será capaz de hacer, facilidad de uso del mismo, disponibilidad y algunas otras restricciones. Este proceso se corresponde con la primera etapa en el modelo en cascada, el cual se explica en el apartado *Metodología*.

En este caso se van a dividir los requisitos en dos tipos: requisitos funcionales y requisitos no funcionales. Más adelante se detallarán las particularidades de cada uno.

Cada uno de estos requisitos vendrá representado por la tabla 6:

(Identificador)	(Nombre)
Descripción	
Entradas	
Proceso	
Salidas	
Prioridad	

Tabla 6: Plantilla requisitos

A continuación, se explica en más detalle el significado de cada uno de los campos que componen esta tabla:

El **identificador** es un código que sirve para diferenciar a cada requisito individualmente. Tendrá la siguiente forma: XX_NN; donde XX podrá ser RF o RNF dependiendo de si el requisito al que se hace referencia es funcional o no funcional, y donde NN será un número entre 00 y 99.

El **nombre**, del cual no hace falta explicar nada.

La **descripción** contendrá una frase con la que se intentará resumir el propósito del requisito.

En las **entradas** se especificará lo necesario para llevar a cabo el proceso, como por ejemplo la existencia de algún fichero.

En el **proceso** se describirán en detalle las actividades que se realizarán para comprobar que el requisito ha sido llevado a cabo de forma exitosa. También aporta información técnica sobre cómo ha de realizarse cierta característica del sistema.

En las **salidas** se enumerará lo que se espera obtener de la actividad en cuestión, como por ejemplo que se genere o se actualice un fichero.

Por último, la **prioridad** representa la importancia del requisito en el sistema. Esta prioridad puede tomar tres valores: baja, media o alta.

5.1.1. Requisitos funcionales

En este apartado se van a detallar los requisitos funcionales del sistema especificados para este proyecto.

Los requisitos funcionales definen el comportamiento o los servicios que el programa debe proporcionar. Estos requisitos se complementan con los requisitos no funcionales, los cuales se orientan al diseño y la implementación del sistema.

Requisito funcional 1

RF_01	**Aprendizaje online**
Descripción	El sistema debe ser capaz de realizar aprendizaje online, actualizando la tabla Q y guardando las tuplas de experiencia conforme se realiza una simulación de tráfico.
Entradas	Parámetro '--onlinelearning' y fichero de la tabla Q ('tablaQ.csv'). Opcionalmente puede existir un fichero de tuplas ('tuplas.csv').
Proceso	El usuario ejecutará el programa con el parámetro '--onlinelearning' e iniciará la simulación. El sistema comprobará que los parámetros sean correctos y que existe el fichero 'tablaQ.csv'. Guardará las tuplas de experiencia en el fichero 'tuplas.csv' según se vayan generando y guardará la tabla Q al finalizar la simulación.
Salidas	Ficheros de tuplas de experiencia y tabla Q actualizados. Fichero generado por SUMO que contiene información sobre la simulación.
Prioridad	Alta.

Tabla 7: Requisito funcional 1

Requisito funcional 2

RF_02	Aprendizaje offline
Descripción	El sistema debe ser capaz de realizar aprendizaje offline, actualizando la tabla Q a partir de unas tuplas de experiencia previamente generadas.
Entradas	Parámetro '--offlinelearning', fichero de tabla Q ('tablaQ.csv') y fichero de tuplas de experiencia ('tuplas.csv').
Proceso	El usuario ejecutará el programa con el parámetro '--offlinelearning' e iniciará la simulación. El sistema comprobará que los parámetros sean correctos y que existen tanto el fichero de tuplas como el fichero de la tabla Q. Por último, actualizará la tabla Q con las tuplas de experiencia contenidas en el correspondiente fichero.
Salidas	Fichero de tabla Q ('tablaQ.csv') actualizado y fichero generado por SUMO que contiene información sobre la simulación.
Prioridad	Alta.

Tabla 8: Requisito funcional 2

Requisito funcional 3

RF_03	Ejecución sin aprendizaje
Descripción	El programa podrá ser ejecutado sin aprendizaje online ni aprendizaje offline, en cuyo caso cargará la tabla Q y utilizará esta información en la simulación.
Entradas	Fichero de la tabla Q ('tablaQ.csv').
Proceso	El usuario ejecutará el programa sin parámetros e iniciará la simulación. El sistema comprobará que no existen parámetros y que existe el fichero de la tabla Q. Cargará la tabla Q y realizará una simulación con esta información.
Salidas	Fichero generado por SUMO que contiene información sobre la simulación.
Prioridad	Alta

Tabla 9: Requisito funcional 3

Requisito funcional 4

RF_04	Guardar tabla Q en fichero
Descripción	El sistema podrá guardar la tabla Q en un fichero.
Entradas	Parámetros '--offlinelearning' y/o '--onlinelearning'. Fichero de la tabla Q ('tablaQ.csv').
Proceso	El usuario ejecutará el programa con los parámetros de aprendizaje online y/o aprendizaje offline. Al finalizar cada uno de estos aprendizajes, el sistema actualizará el fichero de la tabla Q.
Salidas	Fichero de la tabla Q actualizado.
Prioridad	Alta.

Tabla 10: Requisito funcional 4

Requisito funcional 5

RF_05	Cargar tabla Q desde fichero
Descripción	El sistema podrá cargar la tabla Q desde fichero.
Entradas	Fichero de la tabla Q ('tablaQ.csv'). Opcionalmente, parámetros '--offlinelearning' y '--onlinelearning'.
Proceso	El usuario ejecutará el programa con los parámetros deseados. En cualquier caso, ya sea una ejecución normal o de aprendizaje, el sistema cargará el fichero de la tabla Q al principio de la simulación, para utilizarla en la simulación o para actualizarla con tuplas de experiencia.
Salidas	Ninguna.
Prioridad	Alta.

Tabla 11: Requisito funcional 5

Requisito funcional 6

RF_06	Guardar tuplas de experiencia en fichero
Descripción	El sistema podrá guardar tuplas de experiencia en un fichero.

Entradas	Parámetros '--offlinelearning' y/o '--onlinelearning'. Opcionalmente, un fichero de tuplas de experiencia previo ('tuplas.csv').
Proceso	El usuario ejecutará el programa con los parámetros de aprendizaje que desee, aprendizaje online y/o aprendizaje offline, e iniciará la simulación. El sistema guardará todas las tuplas de experiencia generadas durante la simulación en el fichero de tuplas correspondiente, añadiendo cada tupla al final del fichero.
Salidas	Fichero de tuplas de experiencia actualizado.
Prioridad	Alta.

Tabla 12: Requisito funcional 6

Requisito funcional 7

RF_07	**Cargar tuplas de experiencia desde fichero**
Descripción	El sistema podrá cargar tuplas de experiencia desde un fichero.
Entradas	Parámetro '--offlinelearning'. Fichero de tuplas de experiencia ('tuplas.csv').
Proceso	El usuario ejecutará el programa con el parámetro '--offlinelearning' e iniciará la simulación. Al principio de la misma, el sistema cargará las tuplas de experiencia desde fichero para poder llevar a cabo el aprendizaje offline.
Salidas	Ninguna.
Prioridad	Alta.

Tabla 13: Requisito funcional 7

Requisito funcional 8

RF_08	**Generación automática de tráfico**
Descripción	El sistema será capaz de generar tráfico automáticamente, asignando cierta probabilidad para cada ruta posible, de forma que cada ejecución disponga de un tráfico distinto.
Entradas	Ninguna.

Proceso	El usuario ejecutará el programa con los parámetros que desee e iniciará la simulación. Al principio de cada simulación, el sistema generará un fichero distinto de rutas, especificando toda la información necesaria para crear tráfico en el simulador (identificador del vehículo, tipo de vehículo, color, ruta a seguir y momento en el que aparece el vehículo).
Salidas	Fichero de rutas con la información concerniente al tráfico ('map.rou.xml').
Prioridad	Alta.

Tabla 14: Requisito funcional 8

5.1.2. Requisitos no funcionales

En este apartado se van a detallar los requisitos no funcionales especificados para este proyecto.

Los requisitos no funcionales imponen restricciones en el sistema a desarrollar, indican cómo se debe hacer el mismo (al contrario que los requisitos funcionales, los cuales indican qué debe hacer) y suelen cortar transversalmente a los requisitos funcionales.

Los aspectos referentes al rendimiento, la disponibilidad, usabilidad y escalabilidad del sistema, entre otros, se recogen dentro de estos requisitos.

Requisito no funcional 1

RNF_04	**Disponibilidad del sistema**
Descripción	El sistema debe ofrecer una disponibilidad completa.
Entradas	El sistema.
Proceso	El sistema debe estar siempre disponible a menos que se pierda la electricidad.
Salidas	Ninguna.
Prioridad	Alta.

Tabla 15: Requisito no funcional 1

Requisito no funcional 2

RNF_02	Incluir parámetros para los distintos modos de ejecución
Descripción	El sistema debe ofrecer parámetros para diferenciar qué quiere ejecutarse: aprendizaje online, aprendizaje offline y ejecución sin aprendizaje.
Entradas	El programa.
Proceso	El usuario ejecutará el programa sin parámetros, con el parámetro '--offlinelearning', el parámetro '--onlinelearning' o ambos a la vez.
Salidas	El programa con la opción de incluir parámetros.
Prioridad	Alta.

Tabla 16: Requisito no funcional 2

Requisito no funcional 3

RNF_03	Sistema operativo
Descripción	El sistema debe permitir ser instalado en un sistema operativo Windows y los usuarios podrán ejecutarlo en el mismo.
Entradas	El sistema.
Proceso	El usuario instalará el programa en un sistema operativo Windows con éxito.
Salidas	Ninguna.
Prioridad	Alta.

Tabla 17: Requisito no funcional 3

Requisito no funcional 4

RNF_01	Cantidad de información almacenada
Descripción	El sistema debe permitir la creación de tantas tablas Q y tuplas de experiencia por parte del usuario como sea posible.
Entradas	Infraestructura.

Proceso	El máximo de información que el sistema puede almacenar dependerá de la infraestructura del mismo. El sistema deberá soportar tantas tuplas de experiencia y tablas Q como recursos existan.
Salidas	Cantidad de información almacenada.
Prioridad	Alta.

Tabla 18: Requisito no funcional 4

5.2. Diseño del software

En este apartado se va a explicar cómo se ha diseñado el sistema, qué componentes lo forman y cuál es la función de cada uno.

Esta etapa se corresponde con la segunda fase del modelo en cascada.

El sistema está diseñado en capas, donde cada capa puede comunicarse con la anterior y la siguiente tal y como se muestra en la ilustración 14:

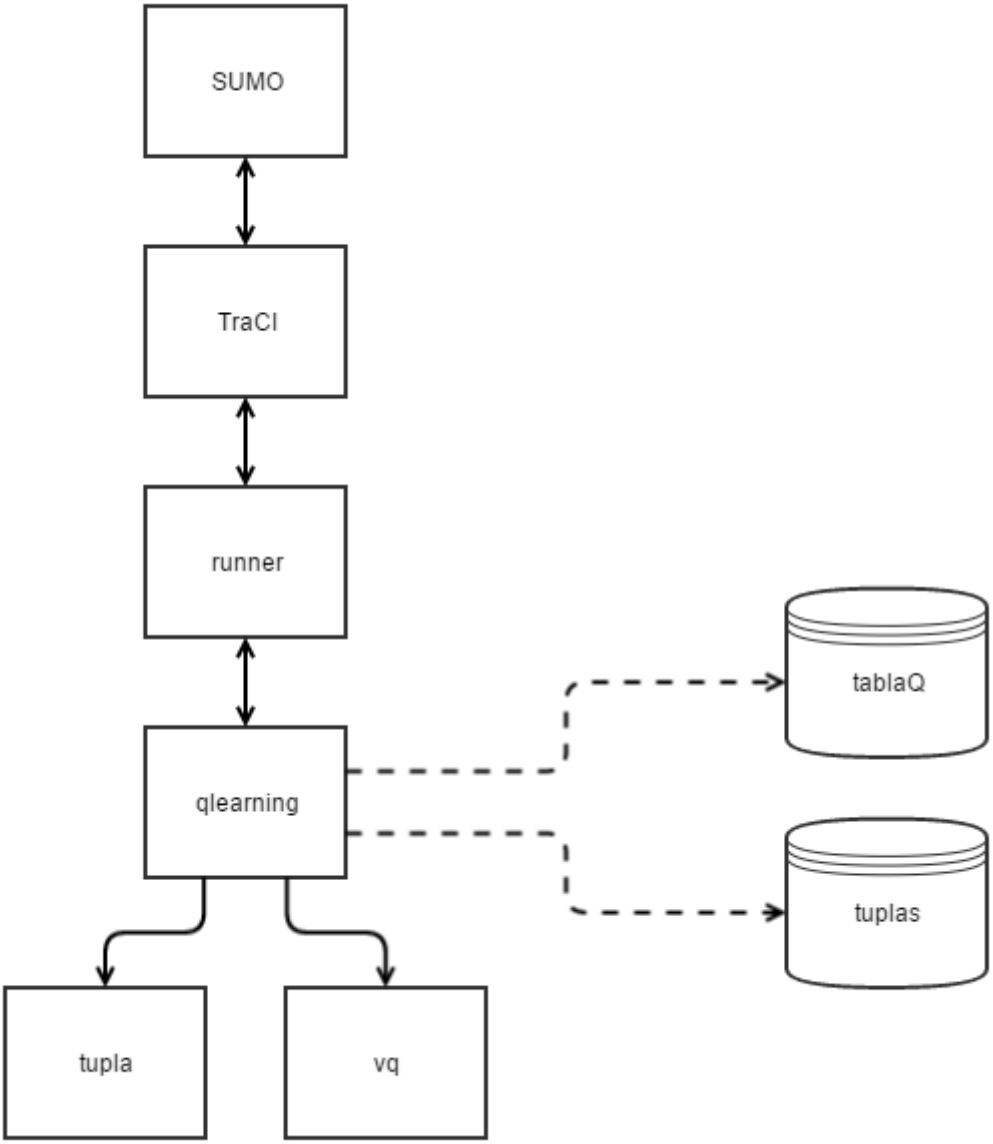

Ilustración 14: Arquitectura del sistema

La capa superior es **SUMO**, el simulador de tráfico utilizado en este proyecto y del que ya se ha hablado en el *Estado del arte*. Éste es un subproceso que se ejecuta y actúa como servidor durante las simulaciones. Es el encargado de mostrar la GUI de la aplicación y realizar todos los cálculos necesarios, aunque para ello abre un puerto (en nuestro caso el 8873) a través del cual nos tenemos que comunicar con el proceso para darle instrucciones (por ejemplo, cuándo empezar y acabar una simulación, cuándo calcular el siguiente paso de una simulación que se esté ejecutando o cambiar el estado de un semáforo) y para obtener toda la información que nos proporciona (por ejemplo, el tiempo de espera de los vehículos, su velocidad o el porcentaje de ocupación de las vías).

Para realizar todas estas comunicaciones con el servidor SUMO, disponemos de la herramienta **TraCI**. Una vez levantado el servidor, nos conectaremos con él desde nuestro proceso cliente con TraCI tal y como se muestra en la ilustración 15:

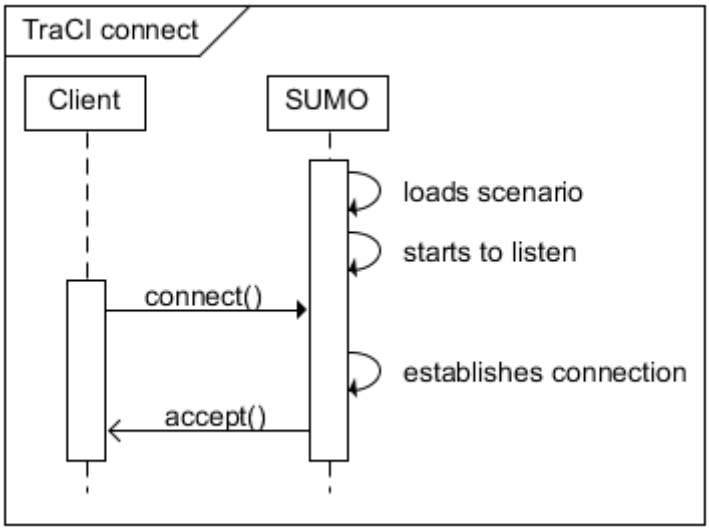

Ilustración 15: Conexión TraCI-SUMO. Extraído de [14]

Una vez el cliente está conectado con SUMO a través de esta conexión, TraCI nos ofrece total control sobre el simulador. Mediante distintas llamadas a funciones podemos iniciar la simulación, avanzar un paso en la misma, parar la simulación, obtener y modificar el estado de los semáforos, obtener la velocidad máxima de una vía, los vehículos que se encuentran en una vía, el número de vehículos parados en un semáforo y el tiempo de espera de los vehículos en un semáforo, entre otras muchas funciones.

En la ilustración 16 se puede ver cómo funciona el mecanismo de desconexión entre TraCI y SUMO:

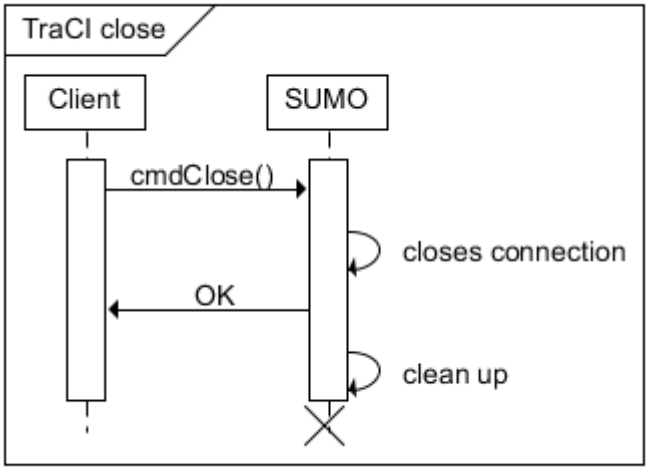

Ilustración 16: Desconexión TraCI-SUMO. Extraído de [14]

El siguiente componente del sistema es **runner**. Éste es el encargado de manejar la simulación y el proceso de aprendizaje. Es el que realiza las llamadas a TraCI, el que comprueba los parámetros de entrada del programa, el que obtiene la información necesaria del simulador y realiza un pre-procesamiento de los datos para pasárselos después a los siguientes componentes, que son los que se encargan del aprendizaje.

Qlearning es el componente donde se realiza el aprendizaje como tal. Recibe de las capas superiores información del simulador y utiliza esta información para actualizar la tabla Q. Contiene también funciones para elegir la mejor acción siempre o seguir una estrategia ε-greedy, además de funciones para modificar los valores alpha y épsilon del algoritmo.

Es capaz de guardar la información recibida de las capas superiores en forma de tuplas en disco. Así mismo, también es capaz de guardar la tabla Q en fichero y cargarla en posteriores simulaciones. Esto, unido a la posibilidad de cargar también las tuplas, otorga al sistema la posibilidad de realizar aprendizaje offline a partir de la información guardada.

Este componente utiliza **tuplas** para trabajar. El elemento del mismo nombre que está subordinado al Qlearning no es más que la abstracción de estas tuplas: una clase que contiene la información relativa a las mismas (estado, estado siguiente, acción y refuerzo), además de algunas funciones útiles para acceder a esta información.

Por último, el componente **vq** (vector quantization) ofrece al módulo Qlearning la posibilidad de cuantificar estados y acciones. Esto es, dado un estado o acción cualquiera, determina a qué estado o acción del modelo diseñado pertenece el estado dado. Es un componente altamente reutilizable y cuya utilidad reside principalmente en algoritmos de clustering.

Este diseño por capas permite de forma sencilla cumplir con todos los requisitos establecidos en la etapa anterior, además de facilitar la implementación del mismo y aumentar la reusabilidad de muchos de estos componentes en futuros proyectos.

5.3. Descripción del espacio de estados

Un estado debe describir las características más importantes, desde el punto de vista del trabajo, del entorno en el que está sumergido el agente en un momento concreto. Estos atributos representarán el entorno y dependiendo de las características elegidas, esta representación será de mayor o menor fiabilidad.

La interfaz TraCI nos ofrece la posibilidad de obtener varias métricas en cada paso de la simulación. Algunas de éstas son:

- La **velocidad media** en un carril. La velocidad media representa la fluidez del tráfico de forma fiable. A mayor velocidad media, se puede asegurar que existe una mayor fluidez del tráfico. En el caso de que no exista ningún coche en el carril, se devolverá la velocidad máxima de la vía. Así, esta métrica resulta de gran importancia en el ámbito del trabajo, cuyo objetivo es reducir los tiempos de espera y, por consiguiente, mejorar la fluidez del tráfico.

- El **número de vehículos** en un carril. Es una métrica fundamental para describir el estado. Ha sido utilizada en la mayoría de los trabajos expuestos en el apartado *Estado del arte*. Detalla de manera clara y sencilla la ocupación de un carril. Además, es sencillo obtener esta información en el mundo real con un radar.

- El **porcentaje de ocupación** de un carril. Es una característica que intenta describir la misma información sobre el estado que la anterior, pero ésta tiene un inconveniente: cuando se tratan vías de mayor y menor longitud, a pesar de existir el mismo número de vehículos en estas vías, aquellas que sean de mayor longitud obtendrán un menor porcentaje de ocupación, con lo que parecería que en estos carriles existe un menor flujo de tráfico, cuando en realidad existe el mismo número de vehículos. En definitiva, el número de vehículos describe con mayor fidelidad el estado que el porcentaje de ocupación.

- El **tiempo de espera** de los vehículos de un carril. Esta métrica representa el tiempo que llevan esperando todos los vehículos de un carril, sumando el tiempo de espera de cada uno de ellos. Esta característica es esencial en el trabajo descrito, pues se quiere minimizar el tiempo de espera en los cruces.

- El **número de vehículos parados** o en movimiento en un carril. Este atributo muestra el número de vehículos parados en un semáforo. Junto al atributo del número de vehículos, se podría saber el número de vehículos en movimiento (número total de vehículos menos el número de vehículos parados). Sin embargo, ambos atributos ofrecen la misma información relativa al flujo del tráfico. Esta métrica ayudaría también a mejorar la fidelidad de la representación del estado sobre la realidad.

TraCI además permite obtener métricas relacionadas con la emisión de gases (como el óxido de nitrógeno (NO_x), el dióxido de carbono (CO_2) o el óxido de carbono (CO)). Sin embargo, estas medidas serían más difíciles de obtener en el mundo real, siendo imposible aislar el tráfico del medio ambiente para obtener una correcta medida de las emisiones de los vehículos.

Todas estas características se podrán utilizar para definir los estados, así como también para definir la función de refuerzo, la cual se describe en el estado *Descripción de la función de refuerzo*.

Al escoger los atributos de los estados, se creará un **espacio de estados**. Este espacio de estados englobará todos los diferentes estados que se pueden generar mediante las diferentes combinaciones de todos los valores posibles que pueden tomar cada uno de los atributos que los componen.

Al tratar con valores continuos o atributos que pueden tomar una gran cantidad de valores, es imprescindible discretizar el dominio de los mismos, ya que si no se realiza esto el espacio de estados generado será demasiado grande para manejarlo. Mediante este proceso de discretización, se consigue una representación más simple del entorno, reduciendo en gran medida el número de estados posibles y haciendo así más manejable el problema. Sin embargo, durante este proceso se introduce un hándicap: según cómo se realice esta discretización, el comportamiento del agente puede variar. Por esta razón, es recomendable realizar pruebas del comportamiento del agente tras variar este proceso.

El espacio de estados y su dimensión repercute de forma considerable en el comportamiento del agente. Para espacios de estados muy pequeños, el agente generalizará demasiado las situaciones del entorno y su comportamiento será muy pobre y sencillo. Por el contrario, para espacios de estados muy grandes, el proceso de aprendizaje será más complejo y duradero, resultando normalmente en una gran parte del espacio de estados inexplorado tras el aprendizaje. Por estas razones, se tiende a reducir en la medida de lo posible el espacio de estados, de forma que el comportamiento del agente no sea demasiado simplista pero tampoco se aumente la complejidad y la duración del periodo de aprendizaje de forma innecesaria.

Tras un proceso mediante el cual se estudió el comportamiento del agente con distintas combinaciones de las métricas expuestas anteriormente, se decidió definir los estados de la siguiente forma:

Estado = (Luz del semáforo, Número de vehículos, Velocidad media)

Luz del semáforo {VERDE, ROJO}. Puede tomar los valores VERDE o ROJO dependiendo de si el semáforo está en verde o rojo.

Número de vehículos {MUY BAJO, BAJO, MEDIO, ALTO}. Se han definido cuatro posibles valores de este atributo dependiendo del número de vehículos en cuestión: MUY BAJO si no hay ningún vehículo; BAJO si hay uno o dos vehículos; MEDIO si hay tres, cuatro o cinco vehículos; y ALTO si hay seis o más vehículos en el carril.

Velocidad media {BAJA, MEDIA, ALTA}. Se han definido tres posibles valores para la velocidad media: BAJA cuando la velocidad media es un 20% o inferior con respecto a la velocidad máxima de la vía; MEDIA cuando este porcentaje es del 20% al 70%; y ALTA cuando este porcentaje es superior al 70%.

Mediante esta descripción de los estados, se han definido veinticuatro estados diferentes y se representa con relativa fidelidad el entorno, escogiendo solamente la información más relevante para este trabajo.

5.4. Descripción del espacio de acciones

El espacio de acciones es mucho más pequeño que el espacio de estados, ya que sólo hay dos acciones posibles: cambiar la luz del semáforo o no hacerlo. Para esto, se necesita saber la luz del semáforo con anterioridad (para cambiarla a verde si está rojo, y viceversa), lo cual no es ningún problema pues se dispone de esta información en el estado.

Desde otro punto de vista, se podrían definir otras dos acciones parecidas a las anteriores: cambiar la luz del semáforo a verde y cambiar la luz del semáforo a rojo. En un primer momento estas acciones pueden parecer una mejor idea ya que, a priori, no necesitan saber el estado en el que se encuentra el semáforo antes de realizar la acción (al contrario que antes). Sin embargo, es necesario cambiar a ámbar siempre que se transite de luz verde a luz roja, y para esto es necesario saber desde qué luz se transita. Por esto, estas acciones no suponen ninguna mejora sobre las anteriores.

Así, las acciones definidas son sólo dos: **cambiar luz** del semáforo y **no cambiar luz** del semáforo.

5.5. Descripción de la función de refuerzo

La función de refuerzo es la que determinará la recompensa del agente por las acciones tomadas. Este refuerzo será 1 cuando se tomen acciones beneficiosas y -1 cuando se perjudique el objetivo del sistema.

Como el objetivo del trabajo es mejorar la fluidez del tráfico, el refuerzo deberá reflejar cómo ha influido una acción en la misma. Esto se puede medir mediante distintas métricas, como ya se ha visto anteriormente.

Se podría utilizar el **número de vehículos** e intentar reducir el número de los mismos en un carril. En este caso, el sistema podría recibir refuerzos negativos simplemente por la aparición de vehículos en un carril, por lo que se ha desestimado esta idea.

También se podría medir la **velocidad media** e intentar aumentar la misma. Sin embargo, esto presenta una principal desventaja: habría que adaptar el refuerzo a la velocidad de cada vía para que vías con una mayor velocidad máxima no reciban un mayor refuerzo. Además, existe otra desventaja: incluso después de abrir un semáforo colapsado, la velocidad media de la vía

será muy pequeña, ya que los vehículos empiezan a coger velocidad una vez pasado el semáforo y el refuerzo no sería el adecuado. Por estas razones, no parece que esta métrica sea la ideal.

Por último, se ha considerado utilizar el **tiempo de espera** de los coches en un semáforo como métrica para la función de refuerzo. Esta métrica carece de las desventajas de las anteriores y persigue el objetivo de mejorar la fluidez del tráfico reduciendo los tiempos de espera.

Para poder usar el tiempo de espera como refuerzo, se tiene que normalizar. Para normalizar el tiempo, se usará un tiempo máximo que sea admisible que esperen los coches. El tiempo máximo establecido ha sido 200. Cabe destacar que el tiempo de espera es la suma de todos los coches esperando. Así, se alcanzará el límite establecido cuando, por ejemplo, un coche haya esperado 200 segundos o cuando 10 coches hayan esperado 20 segundos.

Con este proceso, se dispondrá de una variable entre 0 y 1, siendo 0 lo ideal ya que no hay tiempo de espera. Para transformar este dominio entre -1 y 1 se ha aplicado la fórmula 3:

Ecuación 3: Función de refuerzo

$$R = (-1 \cdot \frac{(t_e - 0.5)}{(1 - 0.5)})$$

Donde R representa el refuerzo y t_e representa el tiempo de espera con un valor entre 0 y 1. De esta forma, se obtiene un refuerzo de -1 cuando el tiempo de espera se aproxime a 1, es decir, al máximo establecido, y un refuerzo de 1 cuando el tiempo de espera sea 0.

5.6. Descripción de los parámetros del Q-Learning

Una vez se han definido los estados, las acciones y la función de refuerzo a utilizar por el Q-Learning, se deben ajustar los parámetros del aprendizaje. Estos parámetros, como se puede ver en la ecuación 1, son dos: la **razón de aprendizaje** (α) y el **factor de descuento** (γ).

Al modificar estos parámetros, variará el comportamiento del agente. Los valores ideales para la razón de aprendizaje y el factor de descuento cambian según el problema a resolver. No existe ningún método para encontrar los valores óptimos dado un problema, por lo que generalmente se utiliza ensayo y error hasta encontrar un α y γ adecuados.

Sin embargo, existen estudios que intentan dar una idea sobre cuáles suelen ser los mejores valores en general. En *Learning Rates for Q-learning* [15], se detalla la precisión del modelo obtenido variando la razón de aprendizaje y el factor de descuento en varios problemas. En este trabajo se observa que los mejores resultados se obtienen para una razón de aprendizaje de 0.85 y un factor de descuento de 0.6, aproximadamente.

En este trabajo, tras realizar varias pruebas modificando α y γ, se va a utilizar una razón de aprendizaje ligeramente inferior a la indicada y un factor de descuento mucho menor, ya que el

objetivo es conseguir un modelo preciso con periodos de aprendizaje largos. La razón de aprendizaje será 0.7 y el factor de descuento será 0.2.

6. Experimentación y resultados

En este apartado se va a detallar el proceso de experimentación seguido para probar el sistema propuesto. Primero se va a describir la duración de los periodos de aprendizaje, incluyendo ciclos de exploración, aprendizaje y evaluación. Después se detallarán cada uno de los escenarios elegidos para probar el sistema. Y, por último, se expondrán los resultados obtenidos en cada uno de estos escenarios.

6.1. Descripción de los periodos de aprendizaje

Cada periodo de aprendizaje dispondrá de **10.000 ciclos de simulación**. Cada ciclo de simulación se corresponde con un segundo en la realidad, por lo que cada periodo englobará un total de casi 3 horas de tráfico. Sin embargo, la experimentación es mucho más rápida al poder acelerar este proceso en el simulador.

Durante cada ciclo de simulación se reduce el valor de α, la razón de aprendizaje, y se incrementa el valor de ε, la probabilidad de elegir la mejor acción en vez de una acción al azar.

Al empezar el periodo de aprendizaje con un α muy alto, el modelo sufrirá grandes cambios. A medida que el aprendizaje avanza y α se hace más pequeño, el modelo recibirá ajustes más pequeños y se refinará, obteniendo mejores resultados.

Por otro lado, al incrementar poco a poco ε, al principio se explorarán continuamente nuevos estados del espacio de estados, mientras que al final se escogerán solamente las mejores acciones. Esto permite explorar en cierta medida el espacio de estados sin necesidad de explorar todos, con lo que el aprendizaje es mucho más rápido, lo cual es una de las principales ventajas del Q-Learning.

Así, los **ciclos de exploración** durante los periodos de aprendizaje serán aquellos que tengan un valor de ε menor a uno. Es decir, aún exista la posibilidad de escoger una acción al azar y visitar un estado no visitado anteriormente. En este caso, se han establecido 4.000 ciclos de exploración.

Los **ciclos de aprendizaje** serán aquellos que tengan un valor de α mayor a cero. Es decir, aún se realizan cambios en el modelo. En el sistema desarrollado, esto ocurre hasta los 7.000 ciclos de simulación.

Por último, los **ciclos de evaluación** serán todos aquellos que tengan un α igual a cero y un ε iguales a uno. Es decir, no se explora el espacio de estados y siempre se escoge la mejor acción. De esta forma se puede evaluar el comportamiento del sistema conseguido. Esto ocurre desde el ciclo 7.000 hasta el final de la simulación.

6.1.1. Generación de tráfico

En cada escenario se va a comprobar el comportamiento del sistema ante distintas situaciones de tráfico. Es necesario generar el tráfico a utilizar durante la simulación en base al mapa y a la densidad de tráfico deseada.

Para esto se ha utilizado una herramienta de SUMO llamada *Random Trips* [16]. Esta herramienta permite **generar tráfico de forma aleatoria**. Simplemente hace falta pasarle como parámetro el mapa, la cantidad de ciclos de la simulación y la densidad del tráfico. Además, se pueden especificar también otros atributos como la velocidad de los coches al empezar el recorrido o los tipos de vehículos a generar.

A continuación, se presenta un ejemplo de cómo utilizar esta herramienta para generar un fichero de rutas con toda la información del tráfico generado:

```
randomTrips.py -n map.net.xml -r map.rou.xml -b 0 -e 10000 -p 1 --trip-attributes=
"departLane=\"best\" departSpeed=\"max\" departPos=\"random_free\" type=\"vTypeDist\""
--additional-file .\data\type.add.xml
```

Con el parámetro -n se pasa el fichero con el mapa a utilizar. Con el parámetro -r se especifica el nombre del fichero de rutas a generar. Con los parámetros -b y -e se especifica los ciclos durante los cuales se quiere generar tráfico. Y con el parámetro -p se controla la cantidad de vehículos generados durante cada ciclo de simulación, especificando cada cuántos ciclos de simulación se debe crear un nuevo vehículo.

En los siguientes escenarios se cambiarán estos parámetros para generar distintas situaciones de tráfico con facilidad.

6.2. Descripción de los escenarios

Para comprobar el funcionamiento del sistema, se van a especificar una serie de experimentos que lo someterán a distintas situaciones. En cada una de estas pruebas se cambiará la densidad del tráfico y/o el mapa utilizado en las simulaciones.

Además, se detallará cómo se realizará el aprendizaje y qué métricas se tomarán para medir la eficacia del sistema desarrollado.

6.2.1. Escenario 1 – Mapa real pequeño con poco tráfico

El primer escenario, que se puede ver en la ilustración 17, donde se probará el sistema será un mapa real pequeño (obtenido a través de OpenStreetMap). Se trata de una zona céntrica de Madrid, donde una de las vías más emblemáticas de la ciudad, la Gran Vía, se cruza con otras dos calles más pequeñas.

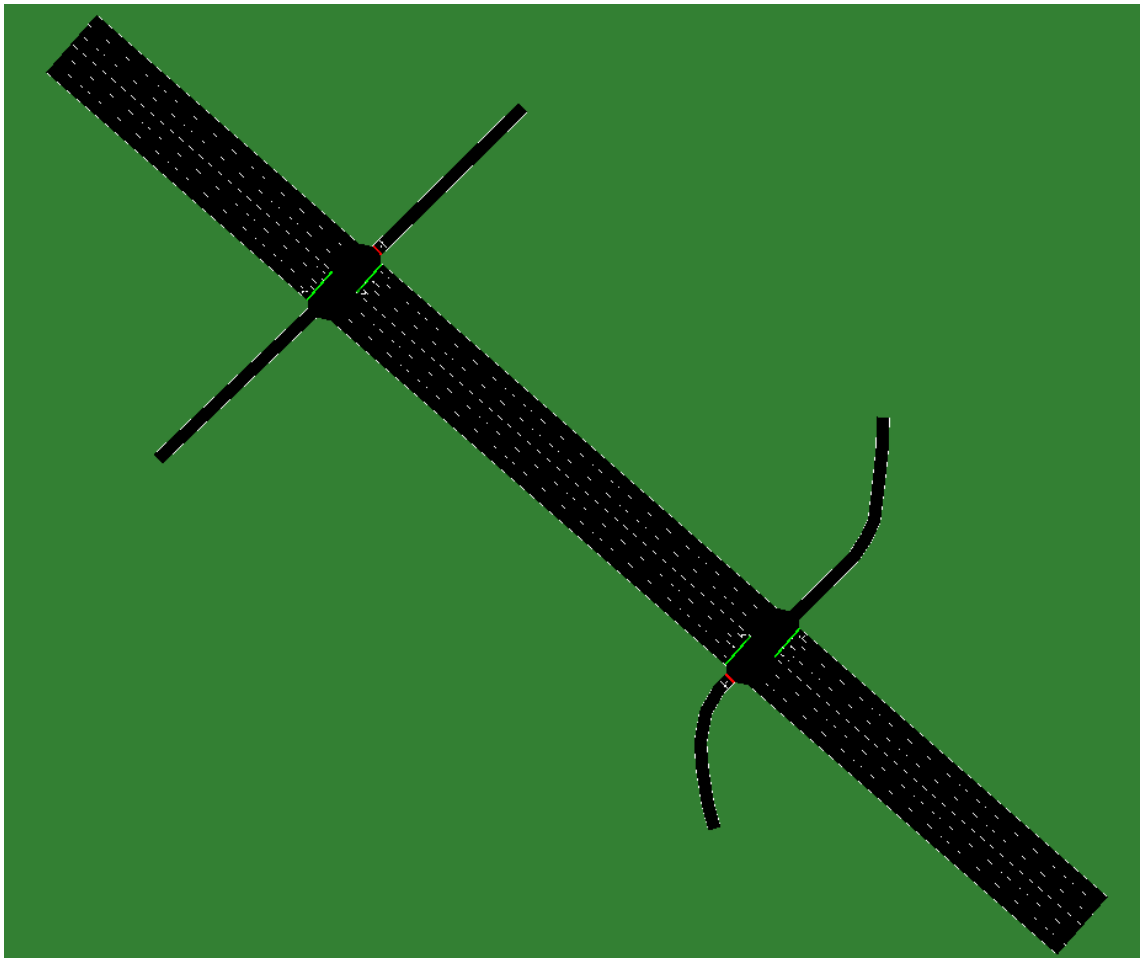

Ilustración 17: Mapa real pequeño

Se eligió este escenario por su relativa sencillez, ya que sólo dispone de dos cruces, y por representar un ejemplo perfecto del objetivo del proyecto: mejorar la gestión de tráfico en grandes ciudades, como Madrid.

En esta prueba se generará un **bajo nivel de tráfico**, creando un sólo vehículo por cada ciclo y medio de simulación (ver el apartado *Generación de tráfico* para más detalles) para comprobar el funcionamiento del sistema en situaciones de tráfico no congestionadas.

Por otra parte, el proceso de aprendizaje puede realizarse de distintas maneras:

- Estableciendo una política fija para todos los semáforos.
- Realizando aprendizaje automático en un semáforo y estableciendo una política fija para el resto.
- Realizando aprendizaje automático para todos los semáforos.

En el primer caso, el aprendizaje no sería el adecuado, ya que ningún semáforo estaría tomando decisiones en base a lo aprendido y no se podrían obtener los refuerzos de dichas acciones.

En el segundo caso, este problema no existiría, ya que un semáforo estaría aprendiendo según las acciones que toma. Además, gracias a la política fija del resto de semáforos, no se crearán congestiones de tráfico debido al mal funcionamiento del sistema en etapas tempranas de aprendizaje.

Por el contrario, en el último caso, debido a las decisiones poco informadas que tomarán los semáforos en etapas tempranas de la simulación, se podrían dar situaciones muy caóticas que dificultarían el aprendizaje.

Por estas razones, se ha considerado que el mejor método de aprendizaje es el segundo, donde solamente un semáforo realiza aprendizaje automático mientras que el resto se rigen por una política fija.

El **aprendizaje** se realizará durante **un episodio**, tal y como se ha descrito en el apartado *Descripción de los periodos de aprendizaje*. Después, se realizará la **evaluación** del sistema, utilizando la tabla Q obtenida tras el aprendizaje, durante **un episodio**, donde todos los semáforos tomarán decisiones basándose en esta tabla Q.

Durante el proceso de aprendizaje, se tomarán medidas locales para comprobar la evolución del sistema en esta etapa. La métrica que se usará será el tiempo acumulado de espera de los vehículos que se encuentran en el carril del semáforo que está realizando el aprendizaje. De esta forma, se podrá observar la mejoría en la toma de decisiones del semáforo inteligente con el paso del tiempo.

Para la evaluación del sistema se tomará el tiempo medio de espera en todos los semáforos y se compararán los resultados del sistema desarrollado con los semáforos de política fija y de política aleatoria. Así, se podrá ver si el sistema se comporta mejor que el peor de los sistemas posibles (decisiones aleatorias) y si se comporta mejor que el sistema actual (política fija).

6.2.2. Escenario 2 – Mapa real pequeño con mucho tráfico

El segundo escenario dispone del mismo mapa que el anterior, pero se generará un **elevado nivel de tráfico**, creando un vehículo por cada 0.75 ciclos de simulación, es decir, existirá más del doble de tráfico que en el anterior ejemplo.

El modo de realizar tanto el aprendizaje como la evaluación del sistema en este escenario es exactamente igual al explicado en el escenario anterior, donde se utilizan algunas métricas para valorar el aprendizaje y para comparar el resultado final con sistemas de política fija y aleatoria.

6.2.3. Escenario 3 – Cuadrícula mediana con poco tráfico

El tercer escenario, que se puede ver en la ilustración 18, se trata de un mapa en cuadrícula. Este mapa es mucho más grande que el anterior y dispone de una gran cantidad de

cruces. Se eligió este mapa ya que su forma se asemeja a las calles de muchas grandes ciudades como Barcelona o Nueva York, donde se construyeron siguiendo este patrón cuadriculado.

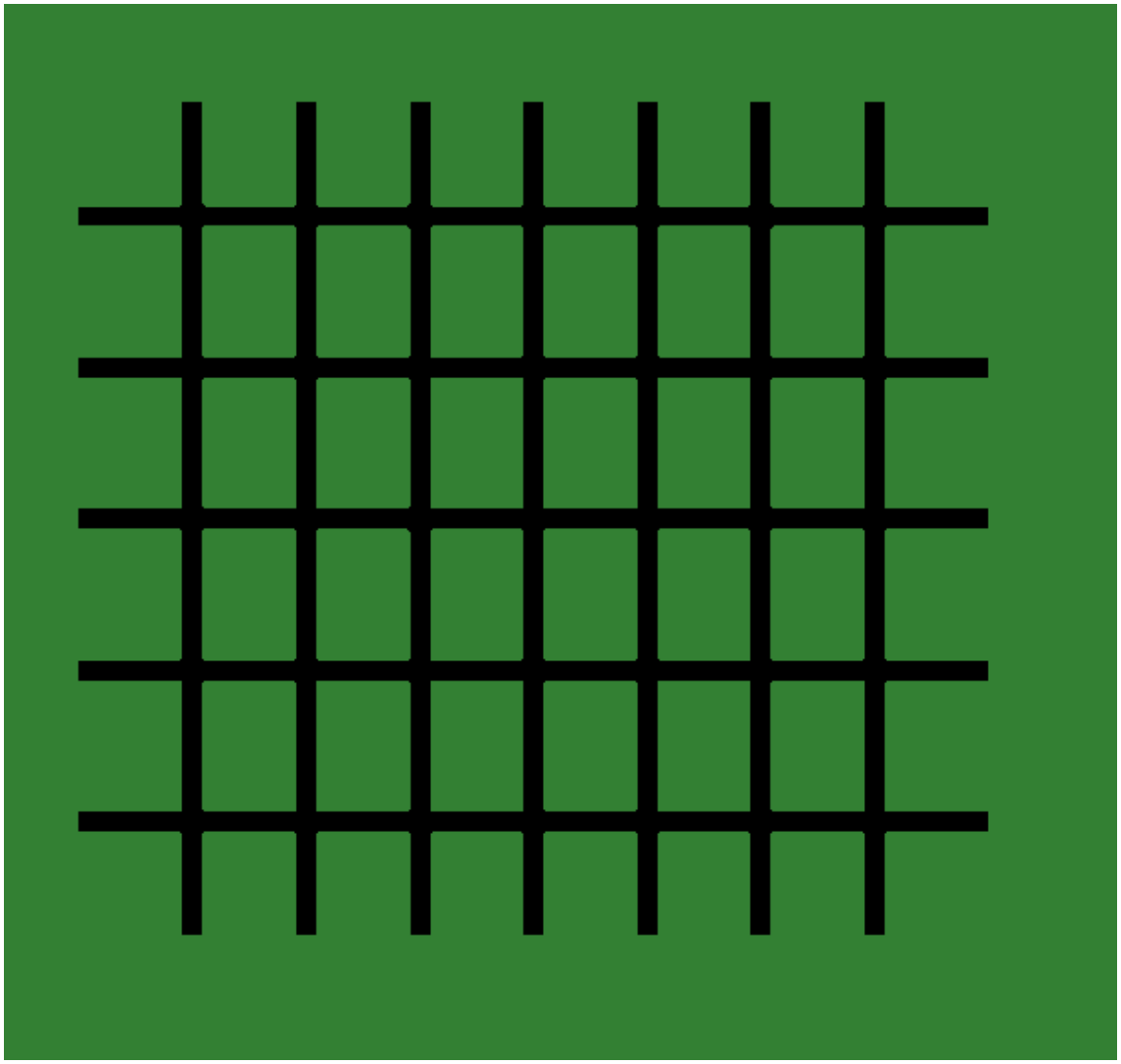

Ilustración 18: Cuadrícula mediana

En este caso se generará un **nivel de tráfico bajo**, creando un sólo vehículo por cada ciclo de simulación, para comprobar el funcionamiento del sistema en este tipo de ciudades y en situaciones de tráfico no congestionadas. En este ejemplo, al tratarse de un mapa mucho más grande, es necesario generar más vehículos para crear una situación de tráfico similar a la del escenario 1.

Tanto el proceso de aprendizaje como el de evaluación del sistema en este escenario es exactamente igual al explicado en el primer escenario. Se estudiará la evolución del aprendizaje y se compararán los resultados finales del comportamiento del sistema con los resultados obtenidos con política fija y aleatoria.

6.2.4. Escenario 4 – Cuadrícula mediana con mucho tráfico

El cuarto escenario dispone del mismo mapa que el anterior, pero se generará un **elevado nivel de tráfico**, creando un vehículo por cada 0.85 ciclos de simulación.

El modo de realizar tanto el aprendizaje como la evaluación del sistema en este escenario es exactamente igual al explicado en el primer escenario, donde se valorará el aprendizaje y se comparará el resultado final con sistemas de política fija y política aleatoria.

6.2.5. Escenario 5 – Cuadrícula grande con poco tráfico

El quinto escenario tiene un mapa distinto, el cual se puede ver en la ilustración 19, pero similar a los dos escenarios anteriores. Se trata también de una cuadrícula, sólo que de mayor tamaño. El objetivo de este escenario es comprobar el funcionamiento del sistema cuando se escala a un elevado número de cruces.

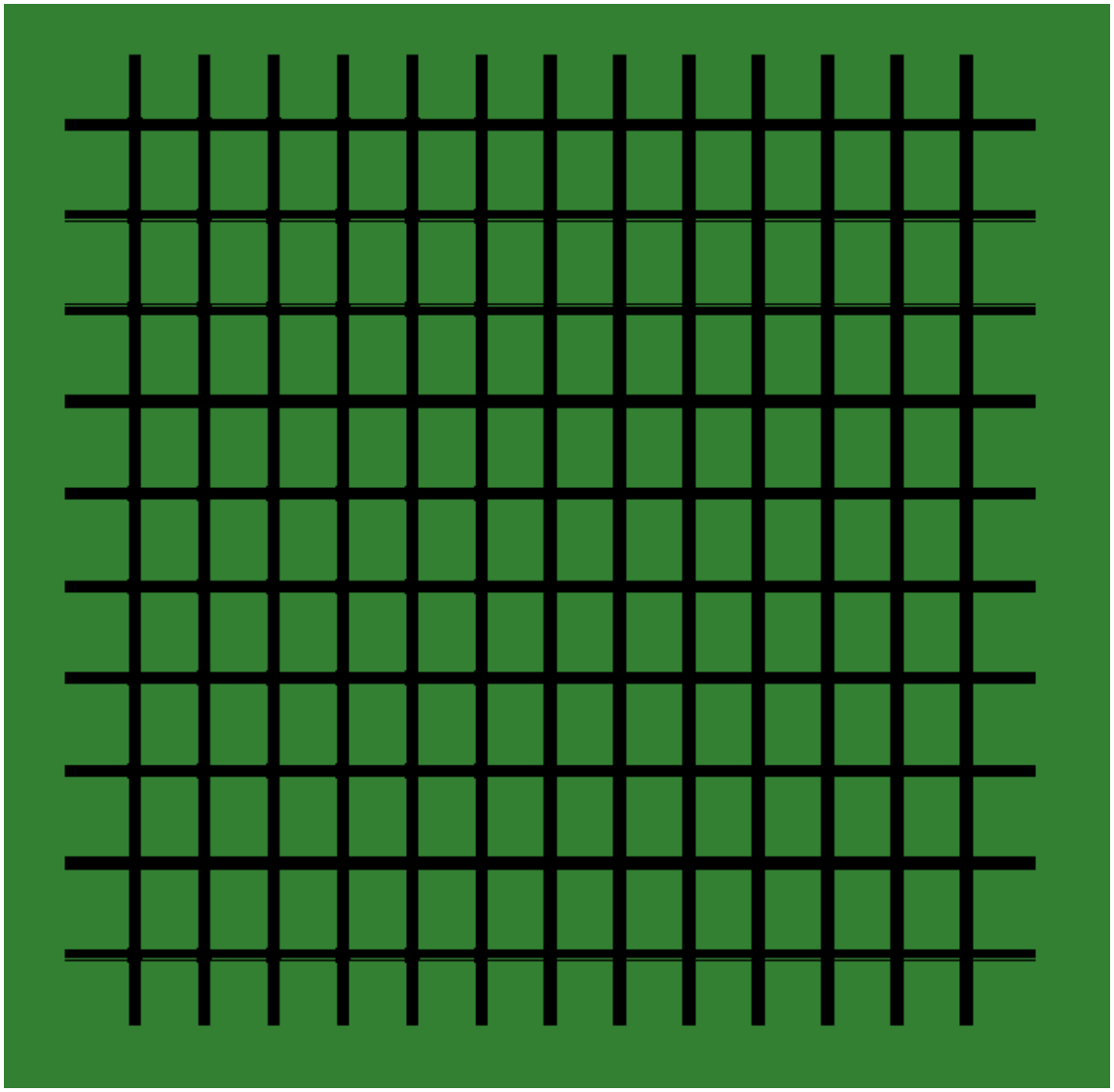

Ilustración 19: Cuadrícula grande

En este caso se generará un **nivel de tráfico bajo**, creando un vehículo por cada 0.65 ciclos de simulación, para comprobar la respuesta del sistema con poco tráfico, pero a una escala mayor.

En este escenario, el aprendizaje y la evaluación de los resultados será similar a los anteriores: se estudiará el aprendizaje del sistema en el tiempo y se compararán los resultados finales con los sistemas de política fija y aleatoria.

6.2.6. Escenario 6 – Cuadrícula grande con mucho tráfico

El sexto y último escenario dispone del mismo mapa que el anterior, pero se generará un **elevado nivel de tráfico**, creando hasta un vehículo por cada 0.35 ciclos de simulación.

Al igual que en los anteriores escenarios, el proceso de aprendizaje y evaluación del sistema en este caso se realizará como se ha explicado ya en el escenario 1: un primer episodio

de aprendizaje donde se podrá ver cómo evoluciona el sistema con el tiempo y un segundo episodio donde se comparará con sistemas de política fija y aleatoria.

6.3. Evaluación de los resultados

En este apartado se van a mostrar los resultados obtenidos al realizar la experimentación en todos los escenarios tal y como se ha descrito en los apartados anteriores. Por cada escenario, se mostrarán los resultados del proceso de aprendizaje y se comparará el comportamiento de los sistemas de política fija y aleatoria con el desarrollado en este proyecto. Por último, se mostrará una tabla que reflejará el tiempo medio de espera de estos tres sistemas en el escenario en cuestión.

6.3.1. Resultados Escenario 1

El primer escenario, el cual se puede ver en la ilustración 17, tiene como objetivo evaluar el comportamiento del sistema en un espacio pequeño donde no exista un gran número de vehículos.

A continuación, se van a describir los resultados del proceso de aprendizaje en este escenario, seguidos a continuación por la comparativa realizada entre los sistemas de política fija, aleatoria y el sistema desarrollado.

Aprendizaje

La ilustración 20 muestra cómo avanza el tiempo de espera de los vehículos que se encuentran en el carril del semáforo en el que estamos realizando el aprendizaje durante la duración de este proceso, es decir, durante los 10.000 ciclos que dura la simulación.

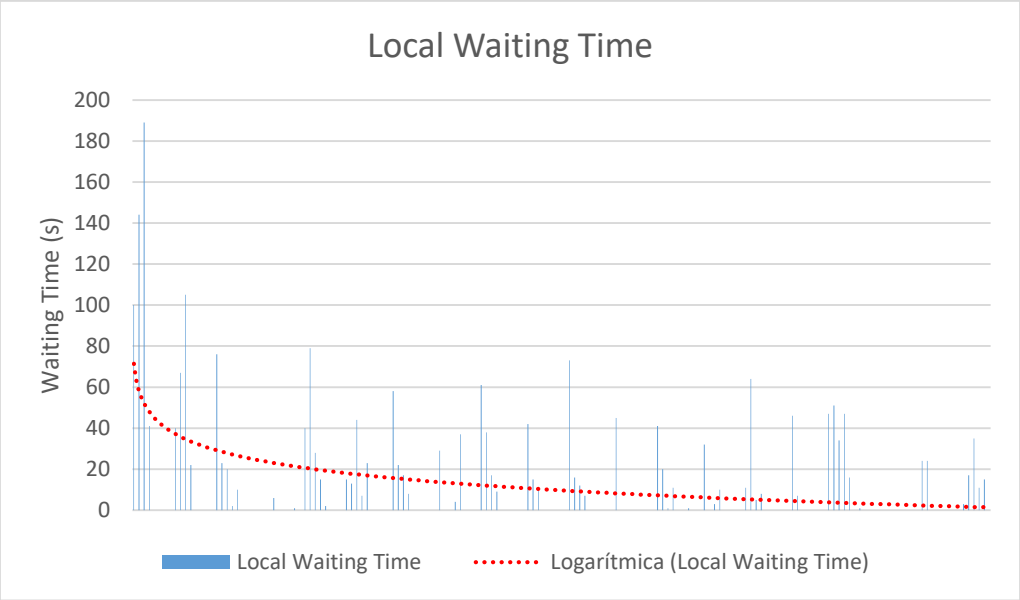

Ilustración 20: Escenario 1: Aprendizaje

Se puede ver claramente que los peores resultados se obtienen al principio del aprendizaje, cuando el agente aún no dispone de la suficiente información como para tomar buenas decisiones. Esto se refleja en grandes tiempos de espera, llegando incluso a alcanzar más de 180 segundos de espera durante los primeros ciclos.

Según avanza el aprendizaje, disminuyen rápidamente estos picos, superando rara vez los 60 segundos de espera durante el resto de la simulación. Así mismo, también se puede ver cómo la media del tiempo de espera tiende a reducirse, llegando a obtener incluso una media de unos 20 segundos aproximadamente durante la fase final del aprendizaje.

Se puede concluir que el aprendizaje ha sido exitoso, ya que el tiempo de espera tiende a reducirse hasta mantenerse durante la última parte del proceso de aprendizaje medianamente constante, en la medida que lo permite la naturaleza irregular del tráfico.

Evaluación

Para comparar los tres sistemas (el de política fija, el de política aleatoria y el desarrollado en este proyecto), se han dividido los resultados de éstos en dos gráficas: una que compara los resultados del Q-learning con política fija y otro que muestra los resultados de la política aleatoria, debido a la gran diferencia en los resultados entre este último sistema y los otros dos.

En la ilustración 21, se puede observar el comportamiento del sistema desarrollado (denominado en la gráfica Q-learning) con respecto al comportamiento del sistema de política fija.

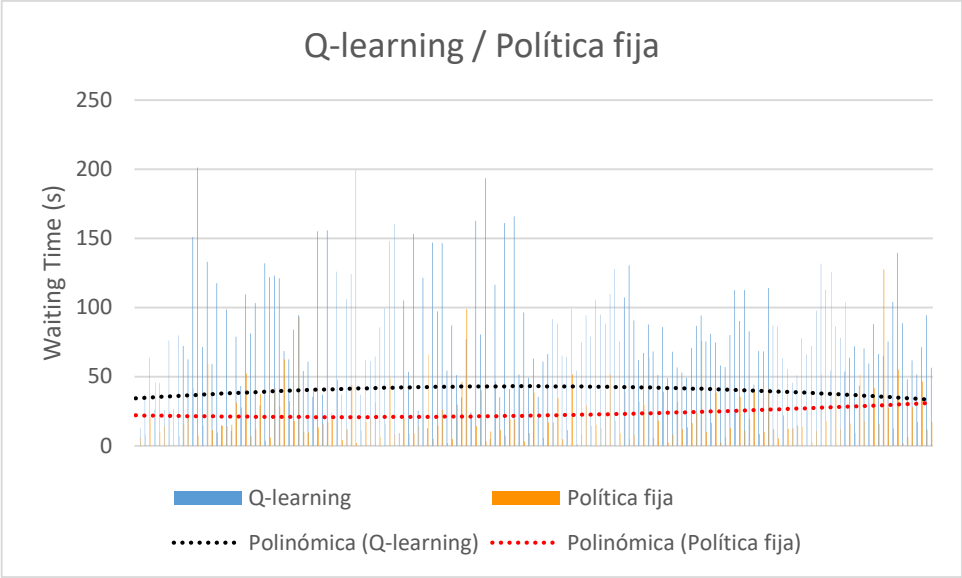

Ilustración 21: Escenario 1: Q-learning / Política fija

El sistema de política fija obtiene resultados ligeramente mejores al Q-learning, manteniendo prácticamente constantemente el tiempo medio de espera por debajo que los tiempos obtenido con el sistema desarrollado. Sin embargo, a medida que se genera más tráfico, en la parte final de la simulación, el tiempo medio de espera de la política fija aumenta, mientras que el del Q-learning disminuye ligeramente.

Por otro lado, el sistema de política aleatoria arroja resultados mucho peores, obteniendo en algunas ocasiones un tiempo de espera superior a los 1000 segundos, como se puede ver en la ilustración 22.

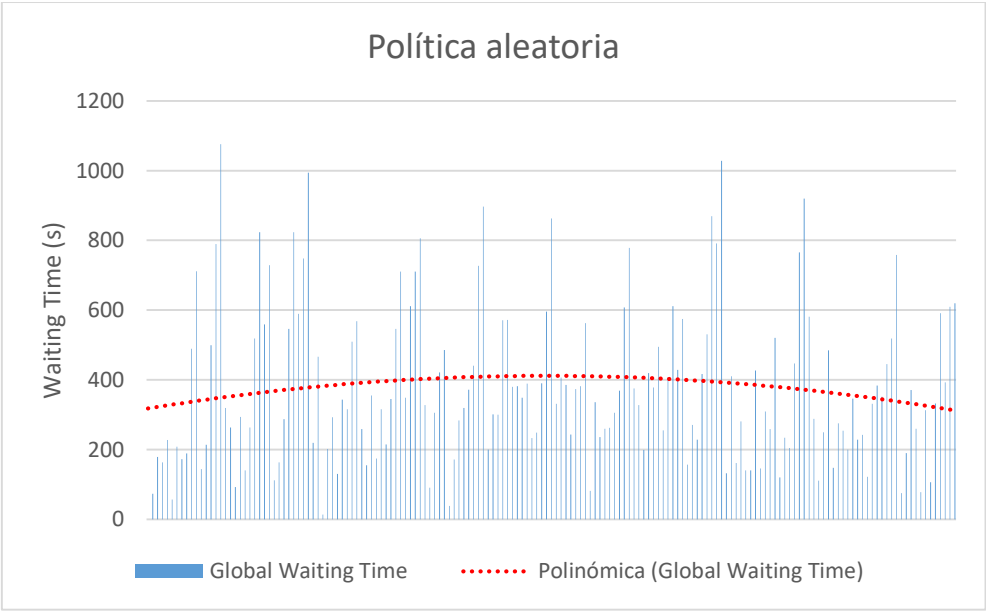

Ilustración 22: Escenario 1: Política aleatoria

El tiempo medio de espera de este sistema se mantiene entre 300 y 400 segundos durante toda la simulación, resultado muy superior al obtenido con los otros dos sistemas, como demuestra la tabla 19, donde se compara el tiempo medio de espera entre estos tres sistemas.

	Tiempo medio de espera (s)
Q-learning	39,94
Política fija	28,43
Política aleatoria	379,09

Tabla 19: Escenario 1: Resultados

Estos resultados demuestran claramente que el sistema de política fija ha obtenido el mejor comportamiento en este caso, donde existía un bajo nivel de tráfico. El sistema desarrollado arroja resultados peores que este sistema, obteniendo un tiempo medio de espera de casi 40 segundos, 10 segundos más que el sistema de política fija. El sistema de política aleatoria es obviamente el peor, ya que su tiempo de espera medio es 10 veces superior al desarrollado en este proyecto.

6.3.2. Resultados Escenario 2

El segundo escenario, el cual dispone del mismo mapa que el primero, posee un alto nivel de tráfico y tiene como objetivo evaluar el comportamiento del sistema en situaciones de tráfico congestionadas.

A continuación, se muestran: primero, los resultados obtenidos durante el aprendizaje en este escenario; y segundo, la comparación del sistema desarrollado con los otros dos sistemas, el de política fija y el de política aleatoria.

Aprendizaje

En la ilustración 23, se puede observar cómo evoluciona el tiempo de espera en el semáforo donde se realiza el aprendizaje durante toda la simulación.

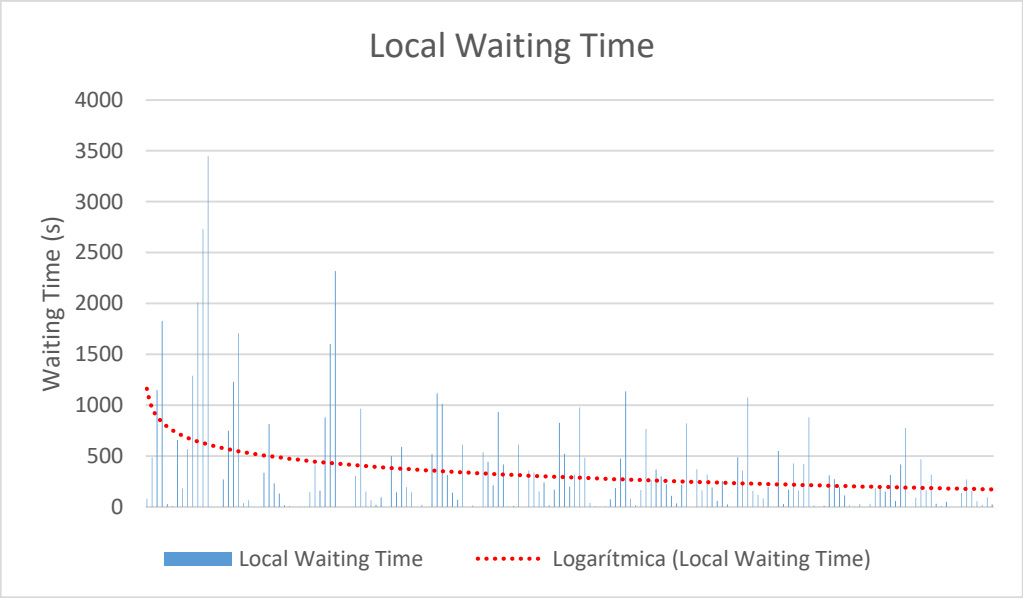

Ilustración 23: Escenario 2: Aprendizaje

Al principio del aprendizaje los tiempos de espera son muy altos, superando en varias ocasiones los 1500 segundos, llegando incluso a alcanzar casi 3500 segundos de espera en una ocasión.

A medida que avanza el proceso de aprendizaje, se nota la mejoría en los tiempos, manteniéndose siempre por debajo de los 1000 segundos, reduciendo la media rápidamente.

En la parte final del aprendizaje, esta mejoría no es tan destacable, ya que los tiempos de espera se mantienen relativamente constantes. Así, se puede concluir que el agente ha realizado un aprendizaje correcto, ya que ha conseguido reducir los tiempos de espera durante este proceso, obteniendo una línea de tendencia similar a la obtenida durante el aprendizaje en el primer escenario.

Evaluación

Al igual que en el primer escenario, se van a presentar los resultados obtenidos con el sistema desarrollado y el sistema de política fija separados de los resultados del sistema de política aleatoria, con el fin de visualizar mejor los mismos.

En la ilustración 24, se muestra cómo, según avanza la simulación, el sistema de política fija obtiene peores resultados, aumentando los tiempos de espera desde unos 200 segundos al principio de la simulación hasta más de 300 segundos al final.

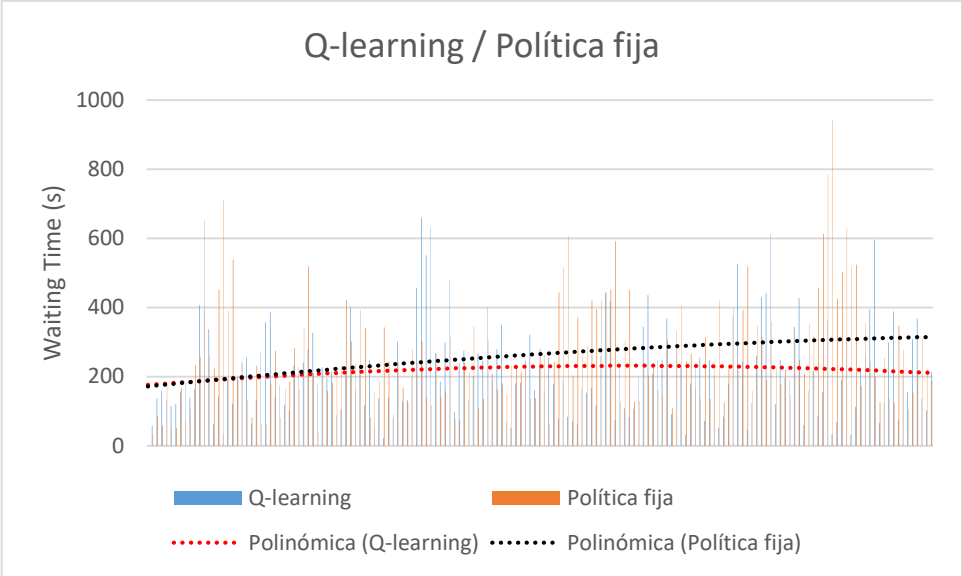

Ilustración 24: Escenario 2: Q-learning / Política fija

Mientras que el sistema desarrollado se mantiene constante alrededor de los 200 segundos de espera durante toda la simulación, permaneciendo siempre por debajo de los resultados obtenidos con el sistema de política fija.

Por el contrario, el sistema de política aleatoria obtiene los resultados que se pueden observar en la ilustración 25.

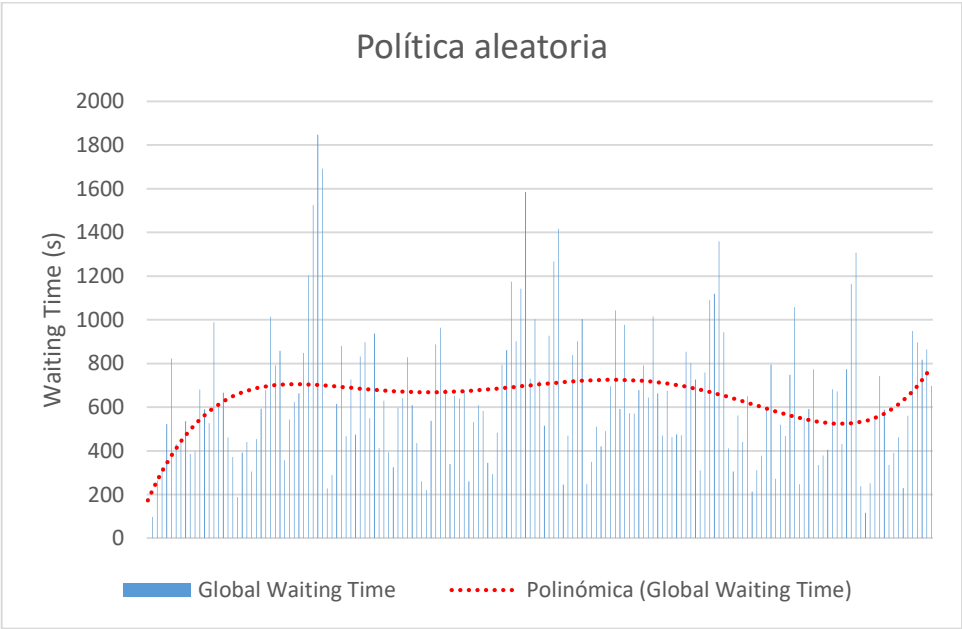

Ilustración 25: Escenario 2: Política aleatoria

En este sistema, se llegan a obtener valores superiores a 1200 segundos en varias ocasiones, y la media se mantiene por encima de los 600 segundos en casi la totalidad de la simulación.

En la tabla 20 se recogen los tiempos medios de espera de los tres sistemas en este escenario, donde se muestra que el sistema desarrollado obtiene una media inferior al de política fija en 40 segundos.

	Tiempo medio de espera (s)
Q-learning	217,24
Política fija	257,99
Política aleatoria	639,87

Tabla 20: Escenario 2: Resultados

Estos resultados demuestran que el sistema desarrollado ha conseguido el mejor comportamiento de los tres sistemas en este escenario, donde existía tráfico congestionado. Mientras que el peor de los tres sigue siendo el de política aleatoria, con un tiempo medio de espera que triplica los resultados de los otros dos sistemas.

6.3.3. Resultados Escenario 3

El tercer escenario, el cual se puede ver en la ilustración 18, es mucho más grande que los anteriores y dispone de un nivel de tráfico moderado. El objetivo de este escenario es evaluar el comportamiento del sistema desarrollado en un entorno de un tamaño mediano.

A continuación, se muestran los resultados del proceso de aprendizaje y se realiza una comparación entre un sistema de política fija, uno de política aleatoria y el desarrollado en este proyecto.

Aprendizaje

En la ilustración 26, se pueden observar las variaciones del tiempo medio de espera de los vehículos que se encuentran en el semáforo que realiza el aprendizaje durante toda la simulación.

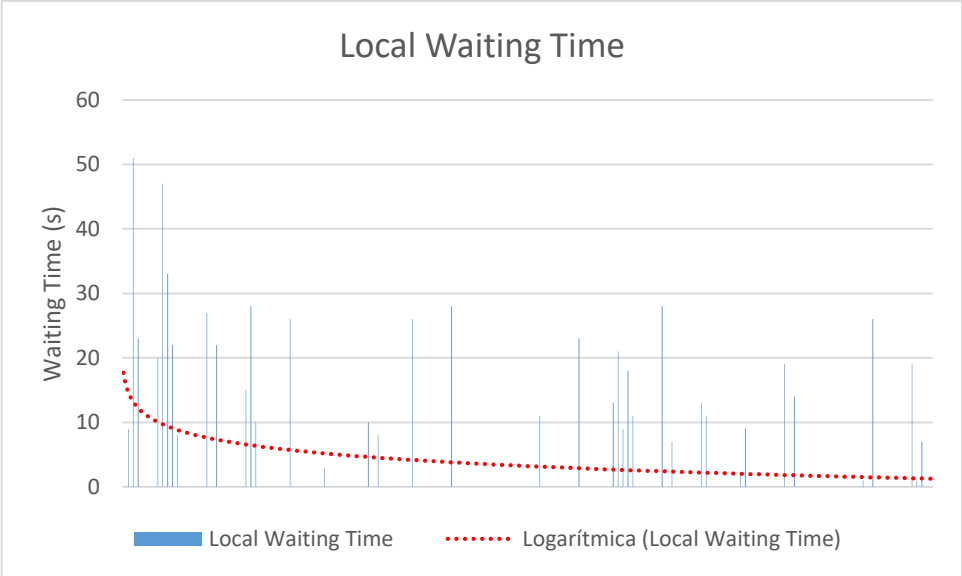

Ilustración 26: Escenario 3: Aprendizaje

En esta gráfica se pueden encontrar grandes rangos vacíos. Esto se debe a la baja afluencia de vehículos, ya que en muchas ocasiones no existe ningún vehículo en el carril que se está evaluando.

También se puede observar que los mayores tiempos de espera se encuentran al principio de la simulación, tal y como ocurre en los dos primeros escenarios, con valores superiores a los 45 segundos. Mientras que en los últimos dos tercios de este proceso los valores se encuentran de forma constante entre los 10 y los 25 segundos, aproximadamente.

Como refleja la línea de tendencia en esta gráfica, el tiempo de espera se reduce rápidamente en la primera fase del aprendizaje y se mantiene casi constante al final del mismo. Por esto, se puede deducir que el aprendizaje se ha realizado con éxito, ya que el agente ha conseguido mejorar los tiempos de espera hasta que éste ha permanecido constante.

Evaluación

Los resultados obtenidos con los tres sistemas se van a representar en distintas gráficas, al igual que se ha hecho con los escenarios anteriores, con el objetivo de mejorar la visibilidad de los mismos.

En la ilustración 27, se comparan los resultados entre el sistema de política fija y el sistema desarrollado, los cuales arrojan resultados bastante diferenciados.

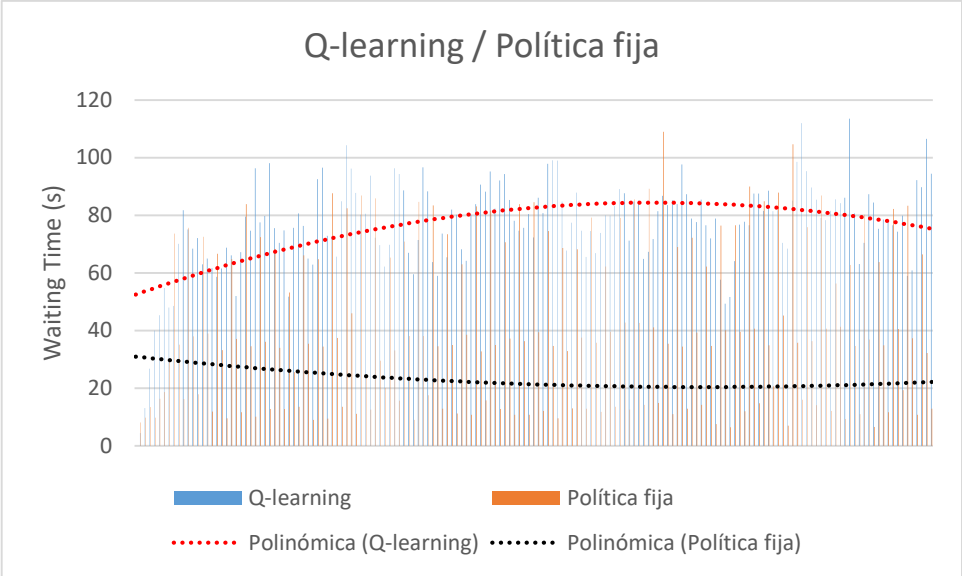

Ilustración 27: Escenario 3: Q-learning / Política fija

En este caso, el sistema de política fija obtiene resultados mucho mejores al sistema desarrollado. Mientras que el tiempo de espera en el Q-learning se mantiene durante prácticamente la totalidad de la simulación entre los 60 y los 100 segundos, en el de política fija estos valores oscilan entre 20 y 80 segundos, aunque suelen encontrarse en su mayor parte valores más cercanos a 20.

Por otra parte, en la ilustración 28 se puede observar el comportamiento del sistema de política fija, cuyos resultados muestran una toma de decisiones mucho peor que en los casos anteriores.

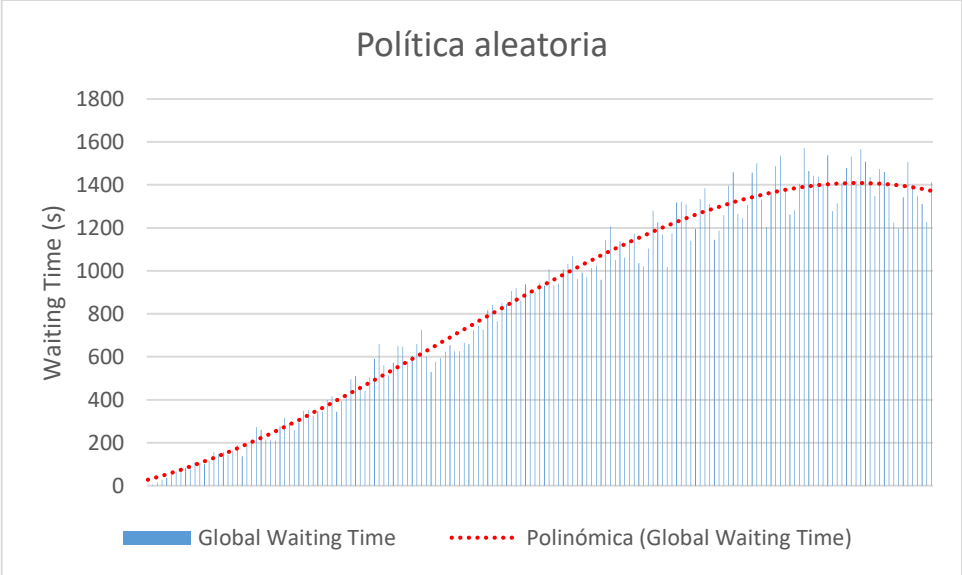

Ilustración 28: Escenario 3: Política aleatoria

Este sistema, como se puede observar por la tendencia creciente en el tiempo de espera durante toda la simulación, consigue crear una gran congestión de tráfico en este escenario, incluso a pesar de no existir un gran número de vehículos.

Estos resultados son mucho peores que los obtenidos con los sistemas de política fija y aleatoria, como se puede consultar en la tabla 21.

	Tiempo medio de espera (s)
Q-learning	60,87
Política fija	23,02
Política aleatoria	848,47

Tabla 21: Escenario 3: Resultados

El sistema desarrollado ha obtenido un tiempo medio de espera de 60 segundos, mientras que el de política fija ha obtenido sólo 23 segundos. Esto, junto con lo comentado sobre la ilustración 27, refleja claramente la superioridad del sistema de política fija en este escenario con baja densidad de tráfico. El sistema de política aleatoria vuelve a ser el peor de los tres sistemas con diferencia.

6.3.4. Resultados Escenario 4

El cuarto escenario dispone del mismo mapa que el anterior, pero existe una mayor afluencia de tráfico. El objetivo aquí es comprobar el comportamiento del sistema desarrollado en situaciones de tráfico congestionado.

A continuación, se muestran los resultados del aprendizaje del sistema, así como la comparación entre este sistema y los de política fija y aleatoria.

Aprendizaje

En la ilustración 29 se pueden observar los tiempos de espera obtenidos durante el periodo de aprendizaje en el carril del semáforo donde se realiza este proceso.

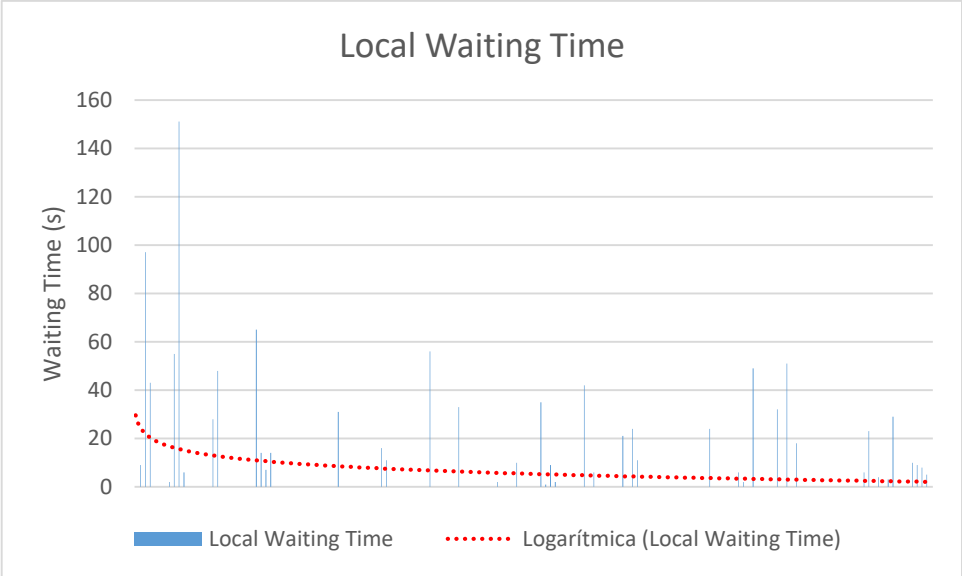

Ilustración 29: Escenario 4: Aprendizaje

Al igual que en los escenarios anteriores, los tiempos de espera al principio de esta simulación son muy superiores al resto, alcanzando valores de 100 y 150 segundos. Después, estos valores se reducen y en la segunda mitad del episodio los tiempos de espera son inferiores a 50 segundos.

La tendencia del tiempo de espera durante este proceso de aprendizaje es similar a las vistas anteriormente, donde existen valores elevados al principio que se van reduciendo a medida que avanza el aprendizaje y acaba manteniéndose constante durante las fases finales.

Evaluación

Para evaluar el sistema desarrollado, se va a comparar primero con el sistema de política fija y luego se mostrarán los resultados obtenidos con el sistema de política aleatoria. En la ilustración 30 se pueden comparar los resultados obtenidos con los dos primeros sistemas mencionados.

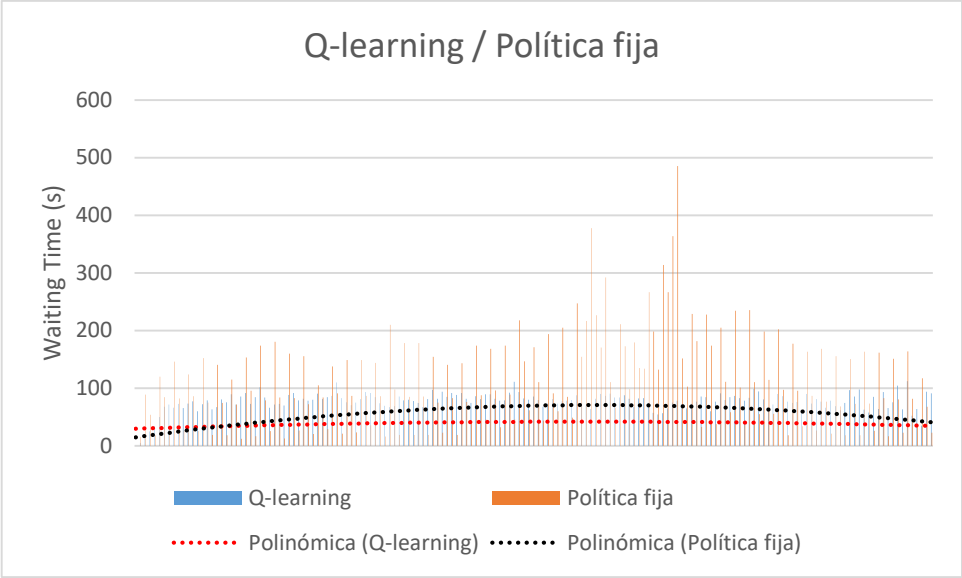

Ilustración 30: Escenario 4: Q-learning / Política fija

En este escenario, el sistema desarrollado mantiene los tiempos de espera sin superar en ningún momento los 100 segundos. El sistema de política fija mantiene los tiempos por debajo de 200 segundos, excepto en un tramo donde los tiempos son más elevados debido a la congestión del tráfico, llegando a superar los 400 segundos de espera.

Por otro lado, el sistema de política aleatoria obtiene resultados muy pobres, acumulando cada vez más tiempo de espera según avanza la simulación, como se puede ver en la ilustración 31.

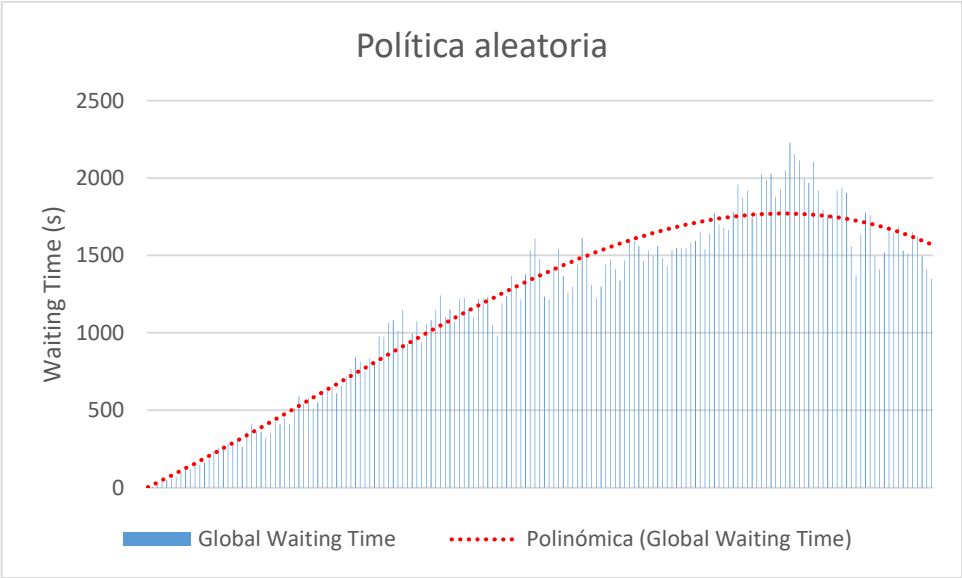

Ilustración 31: Escenario 4: Política aleatoria

Como se puede ver en la tabla 22, el sistema de política aleatoria obtiene un tiempo medio de espera 10 veces superior al de política fija, siendo la media de este último de 111 segundos. El sistema desarrollado obtiene un tiempo medio de sólo 66 segundos.

	Tiempo medio de espera (s)
Q-learning	66,80
Política fija	111,01
Política aleatoria	1172,88

Tabla 22: Escenario 4: Resultados

En conclusión, en este escenario el sistema desarrollado se comporta de forma mucho más óptima que el sistema de política fija y que el de política aleatoria, llegando a reducir incluso a la mitad el tiempo medio de espera con respecto al de política fija.

6.3.5. Resultados Escenario 5

Este escenario dispone del mapa mostrado en la ilustración 19 y un bajo nivel de tráfico. El mapa es similar al anterior, pero es más grande. El objetivo de este escenario es evaluar el comportamiento del sistema desarrollado en entornos de grandes dimensiones.

A continuación, se muestran los resultados del aprendizaje y una evaluación del sistema desarrollado, comparándolo con un sistema de política fija y otro de política aleatoria.

Aprendizaje

En la ilustración 32, se muestra la evolución del tiempo de espera en el semáforo donde se realiza el aprendizaje durante la duración del mismo.

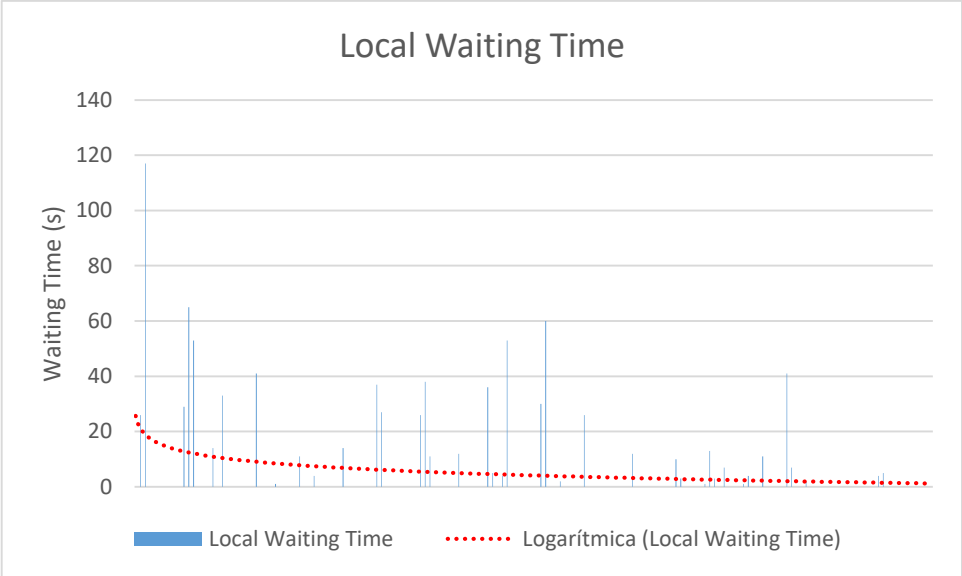

Ilustración 32: Escenario 5: Aprendizaje

Los valores más altos se vuelven a dar una vez más durante los primeros ciclos de aprendizaje, alcanzando casi 120 segundos al comienzo de este proceso. Estos valores bajan hasta encontrarse entre los 30 y los 60 segundos de espera durante la mitad del aprendizaje. En el último tercio del mismo, los valores son especialmente pequeños, destacando solamente en una ocasión un valor de 40 segundos.

La tendencia de la media del tiempo de espera baja a medida que se avanza en el aprendizaje, situación similar a las vistas en los anteriores escenarios. Aunque en esta ocasión la mejoría no es tan notable, se alcanzan valores muy bajos rápidamente y éstos se mantienen constantes al final.

Evaluación

La ilustración 33 presenta los resultados del sistema desarrollado y el sistema de política fija. Se han separado estos dos sistemas del de política aleatoria para que se puedan ver mejor la diferencia entre éstos.

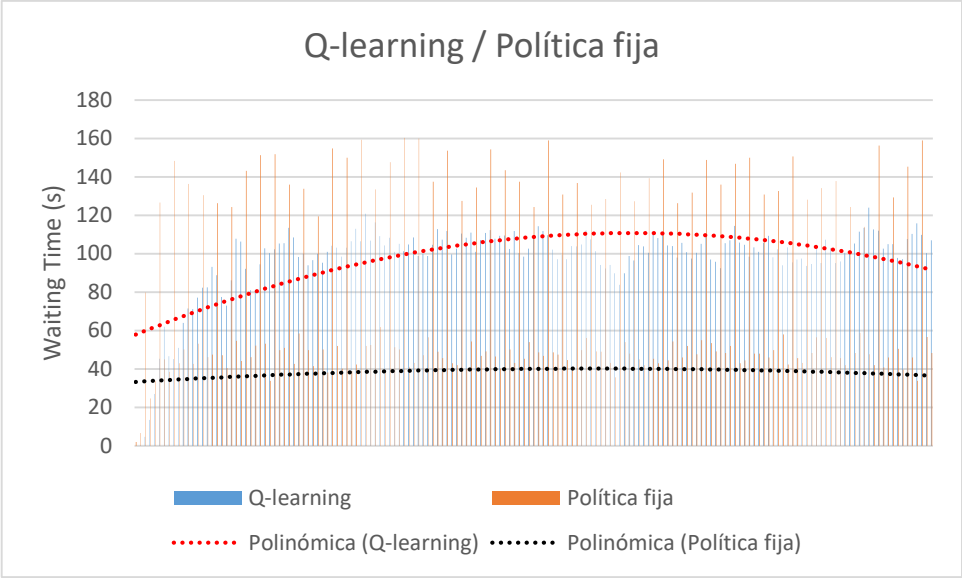

Ilustración 33: Escenario 5: Q-learning / Politica fija

En este caso, el sistema de política fija obtiene los valores más altos, alcanzando constantemente valores por encima de 140 segundos, pero también los más bajos, ya que existe un gran número de ciclos donde el tiempo de espera está entre 40 y 50 segundos. Por su parte, el sistema desarrollado alcanza valores más homogéneos, encontrándose siempre alrededor de los 100 segundos.

Por otro lado, el sistema de política aleatoria arroja los resultados que se pueden ver en la ilustración 34, donde los tiempos de espera son mayores según avanza la simulación, alcanzando al final de la misma los 500 segundos de espera.

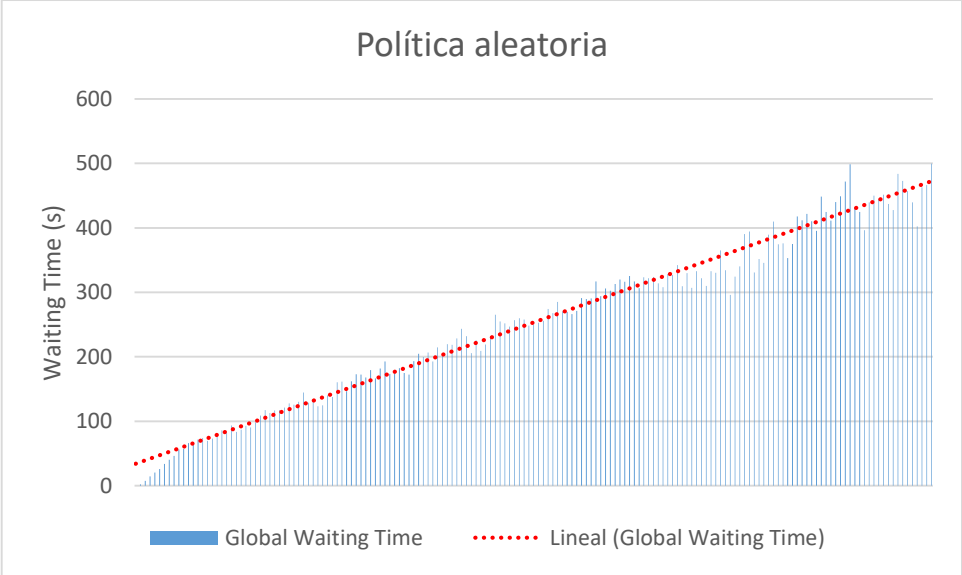

Ilustración 34: Escenario 5: Política aleatoria

La tabla 23 muestra los tiempos medios de espera de cada uno de estos tres sistemas. El de política fija obtiene el mejor tiempo medio con 76 segundos. Después se encuentra el sistema desarrollado que le sigue de cerca con 97 segundos. Por último, el sistema de política aleatoria vuelve a obtener el peor resultado con diferencia con 253 segundos de media.

	Tiempo medio de espera (s)
Q-learning	97,21
Política fija	76,44
Política aleatoria	253,33

Tabla 23: Escenario 5: Resultados

En definitiva, el sistema desarrollado, aunque es el que obtiene los resultados más constantes, rinde ligeramente peor que el sistema de política fija en este escenario de bajo nivel de tráfico.

6.3.6. Resultados Escenario 6

Este escenario es similar al anterior, con la diferencia de que este dispone de un alto nivel de tráfico para evaluar el sistema en entornos grandes y con tráfico congestionado.

A continuación, se presentan los resultados obtenidos tanto durante el proceso de aprendizaje como al evaluar los distintos sistemas con los que se compara el desarrollado.

Aprendizaje

En la ilustración 35 se pueden observar los resultados del aprendizaje en este escenario. Esta gráfica muestra una tendencia similar a los aprendizajes del resto de escenarios: al principio existen tiempos de espera muy altos, superando los 80 segundos de espera en varias ocasiones, y al final estos valores se reducen, manteniéndose casi constantes durante la segunda mitad del proceso entre los 20 y los 40 segundos de espera.

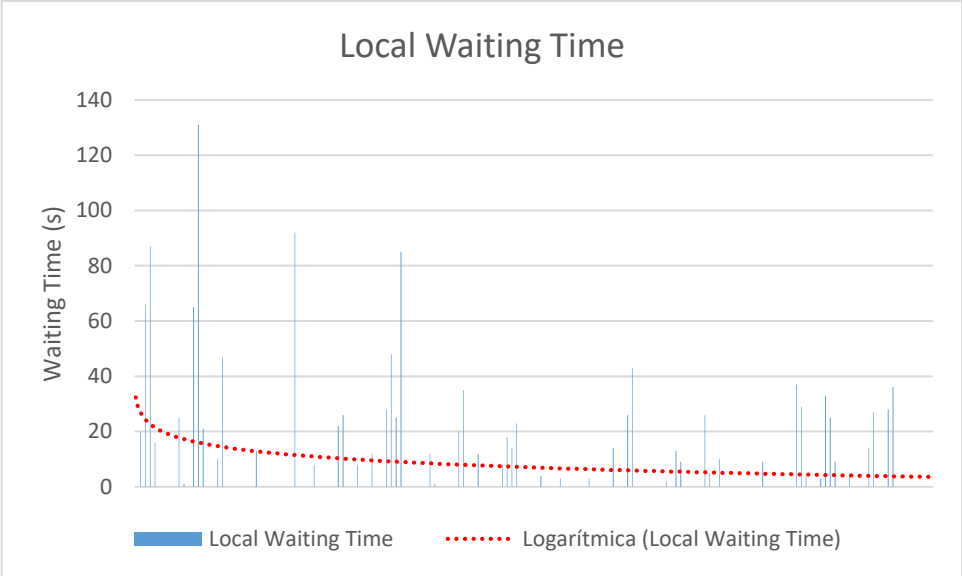

Ilustración 35: Escenario 6: Aprendizaje

Este aprendizaje se puede calificar como bueno, ya que se produce una gran mejoría en el tiempo de espera y éste se mantiene constante en las etapas finales del mismo, al igual que ocurre con el aprendizaje en el resto de escenarios.

Evaluación

En la ilustración 36 se presentan los resultados del sistema de política fija con los del sistema desarrollado, denominado Q-learning. Se han separado estos resultados de los obtenidos con el sistema de política aleatoria para mejorar la visualización de los mismos.

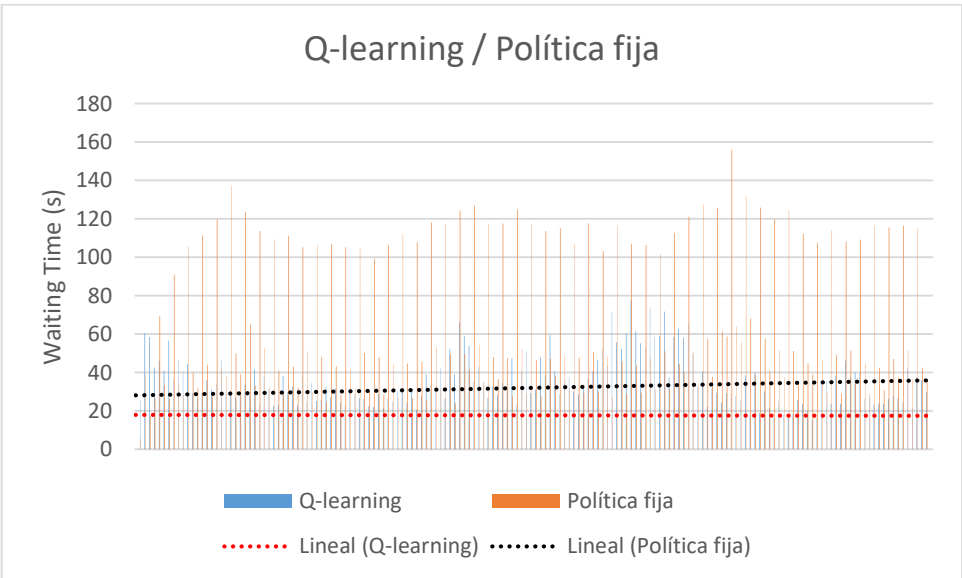

Ilustración 36: Escenario 6: Q-learning / Política fija

Los resultados parecen ser bastante claros en este escenario: el sistema de política fija obtiene valores muy altos, superando constantemente los 100 segundos de espera, mientras que con el sistema desarrollado el tiempo de espera no suele superar los 60 segundos como máximo. Además, como muestran las líneas de tendencia, el sistema de política fija tiende a incrementar el tiempo de espera según avanza la simulación, debido al aumento del tráfico, mientras que el sistema desarrollado se mantiene constante.

Por otro lado, en la ilustración 37 se pueden observar los resultados del sistema de política aleatoria, los cuales son similares a los vistos en los anteriores escenarios: según avanza la simulación, el tiempo de espera aumenta debido a las grandes congestiones de tráfico que se forman por la mala gestión del tráfico.

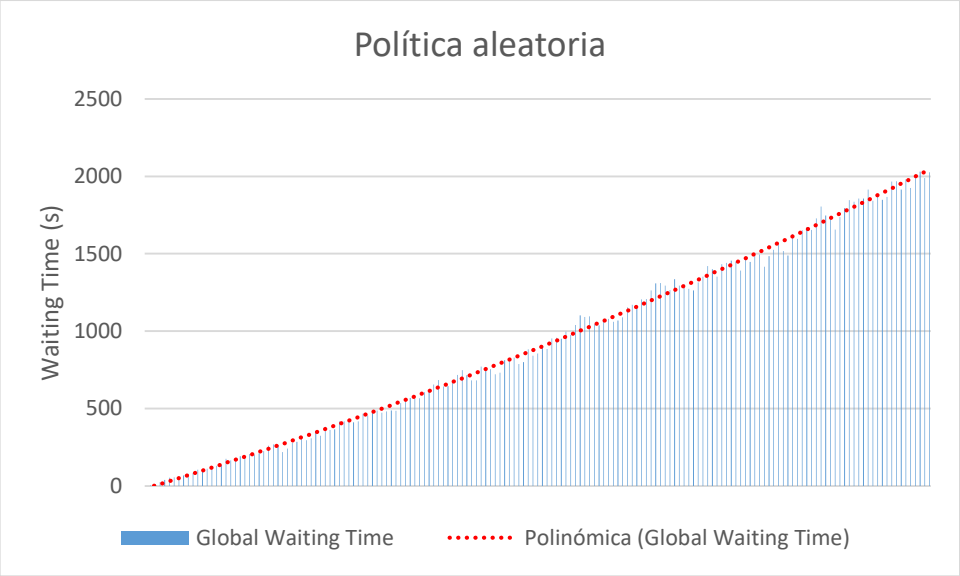

Ilustración 37: Escenario 6: Política aleatoria

En resumen, el sistema desarrollado ha obtenido un tiempo medio de espera de 35 segundos, por los 63 segundos del sistema de política fija y los 931 segundos del de política aleatoria. Estos resultados están recogidos en la tabla 24.

	Tiempo medio de espera (s)
Q-learning	35,44
Política fija	63,78

Política aleatoria	931,64

Tabla 24: Escenario 6: Resultados

En definitiva, en este escenario, donde existía una densidad de tráfico elevada, el sistema desarrollado es el que mejor resultados obtiene con diferencia, reduciendo a casi la mitad el tiempo medio de espera de los vehículos con respecto al sistema de política fija. Por lo contrario, el sistema de política aleatoria es el que peor resultados obtiene.

Como conclusión de la etapa de experimentación realizada, tras haberse analizado el comportamiento del sistema desarrollado y haberlo comparado con sistemas de políticas fija y aleatoria, se llega a la deducción de que el sistema desarrollado ha obtenido los mejores resultados en los escenarios 2, 4 y 6, que son los escenarios donde existía un elevado nivel de tráfico. Por el contrario, el sistema de política fija ha sido el mejor en los escenarios 1, 3 y 5, aquellos donde el tráfico era menor.

Dado que el objetivo del trabajo era desarrollar un sistema que gestionara el tráfico de forma eficiente y redujera las congestiones de tráfico, se puede concluir que este objetivo ha sido completado con éxito a la vista de los resultados de los sistemas en situaciones de tráfico congestionado, donde el sistema desarrollado ha sido superior al resto. Sin embargo, no se ha conseguido mejorar el sistema de política fija en situaciones de tráfico poco congestionadas, por lo que el sistema desarrollado debería de mejorarse antes de instalarlo en la realidad.

7. Conclusiones y líneas futuras

En este trabajo se ha desarrollado un sistema multi-agente distribuido que, mediante aprendizaje por refuerzo, gestiona tráfico gracias a las decisiones que toma cada semáforo de forma individual.

El objetivo del proyecto era desarrollar un sistema utilizando aprendizaje automático y evaluar los resultados con respecto a otros dos sistemas: uno de política fija y otro de política aleatoria.

Para lograr dicho objetivo, se integró Q-learning con un software simulador de tráfico llamado SUMO, donde se pueden hacer pruebas relativamente reales sin comprometer la seguridad de ninguna persona.

Se llevaron a cabo distintos experimentos con SUMO, con el fin de comprobar el comportamiento del sistema desarrollado y compararlo con el sistema que reina en la actualidad: el sistema de política fija. Los resultados de estos experimentos arrojaron estas conclusiones:

- El sistema desarrollado se comporta de forma menos eficiente en situaciones de bajo nivel de tráfico que el sistema de política fija.
- El sistema desarrollado es más eficiente en situaciones de tráfico congestionado que el sistema de política fija.

Como se describió en el *Estado del arte*, las grandes ciudades son aquellas que sufren los mayores problemas de tráfico y son el origen de las congestiones de tráfico que tienen un gran impacto económico y medioambiental en la sociedad. Estos resultados invitan a pensar que es posible instalar el sistema descrito en grandes ciudades y obtener beneficios por ello.

En definitiva, este proyecto presenta la posibilidad de crear semáforos inteligentes que sean independientes, otorgando así la capacidad de implementar un sistema que sea altamente escalable y pueda ser instalado en cualquier lugar.

Sin embargo, este sistema dispone de mucho margen de mejora. Empezando por el diseño de estados y la función de refuerzo realizado, los cuales son relativamente simples, se podrían llevar a cabo pruebas con espacios de estados y funciones de refuerzo un poco más complejos, con el fin de mejorar el comportamiento del sistema final.

Por otro lado, el proceso de aprendizaje dispone de muchos parámetros que podrían modificarse para mejorar los resultados. Se podrían alargar o acortar las distintas fases del aprendizaje, ajustar la razón de aprendizaje y el factor de descuento del Q-learning, o utilizar una política de exploración y explotación distinta a ε-greedy.

Además, se podría aplicar aprendizaje automático al discretizar los atributos de los estados. Para ello, se podría utilizar K-medias y hacer uso del módulo incluido en el sistema llamado vq (vector quantization). Este proceso es fundamental en Q-learning y tiene un gran impacto en el resultado final. Así, al aplicar aprendizaje automático se podría lograr una discretización más adecuada, pudiendo llegar a obtener un comportamiento mucho más eficiente.

La fase de experimentación es otra de las áreas que podría ser trabajada en mayor profundidad en un futuro. Se podrían incluir peatones en las simulaciones, los cuales afectarán al tráfico y el sistema deberá adaptarse a esta nueva circunstancia. También se podrían incluir cruces con distinta morfología, creando escenarios asimétricos y cuyo tráfico sea más denso en algunas zonas y menos denso en otras.

En un futuro también se podría incluir un sistema que evite accidentes en los cruces. El diseño del sistema hace que sea posible que los semáforos de un cruce generen situaciones peligrosas al no disponer cada uno de información sobre el estado del resto de semáforos. Así, se pueden dar situaciones donde todos los semáforos están abiertos, dando lugar a posibles accidentes de tráfico. Por esto, se sugiere la inclusión de un componente en el sistema que regule estas situaciones, aunque esto pueda afectar a la escalabilidad del sistema.

Estos son sólo algunos ejemplos de cómo podría seguirse mejorando el sistema desarrollado, tomando éste como base para otros proyectos que se centren en realizar alguna de las tareas mencionadas, sin preocuparse sobre la integración de Q-learning en SUMO.

Marco legal

El software utilizado para este proyecto está sujeto a la licencia GNU General Public License (GPL) que dota al usuario de la libertad de usarlo, estudiarlo, compartirlo y modificarlo. Sin embargo, cualquier modificación que se realice sobre este software, deberá ser distribuido bajo la misma licencia.

Por otro lado, el sistema planteado requiere del uso de cámaras y radares de velocidad. La legalidad de este sistema dependerá de lo que dicten las leyes sobre el uso de estos aparatos en el país o estado donde se quiera instalar.

Por ejemplo, en la página web del *Insurance Institute for Highway Safety* [17] está disponible información legal sobre el uso de cámaras y radares de tráfico en los distintos estados de Estados Unidos. En Nueva Jersey estos mecanismos están prohibidos, mientras que en Arizona están permitidos.

Este es sólo un ejemplo de cómo varía la legalidad de estos aparatos según el lugar donde nos encontremos. Por ende, el sistema desarrollado será legal en algunos lugares e ilegal en otros, quedando a merced de las leyes vigentes en dichas localidades.

Bibliografía

[1] Centre for Economics and Business Research. *The future economic and environmental costs of gridlock in 2030*. Londres, 2014 [en línea]. Disponible en: http://inrix.com/wp-content/uploads/2015/08/Whitepaper_Cebr-Cost-of-Congestion.pdf. [consulta: 04 agosto 2016].

[2] L.P Kaebling, M.L. Littman y A.W. Moore. "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, nº. 4, 237-285, mayo 1996.

[3] C. Watkins y P. Dayan. "Q-Learning: Technical Note", *Machine Learning*, nº. 8, 279-292, 1992.

[4] C. Li y S. Shinamoto. "ETC Assisted Traffic Light Control Scheme for Reducing Vehicles CO2 Emissions", *International Journal of Managing Information Technology*, Vol. 4, nº. 2, mayo 2012.

[5] D. Srinivasan, M. C. Choy y R. L. Cheu. "Neural Networks for Real-Time Traffic Signal Control", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 7, nº. 3, septiembre 2006.

[6] G. Jansson. *Traffic Control with Standard Genetic Algorithm*. Tesis doctoral. Gothenburg: Universidad de Chalmers, Departamento de Tecnologías aplicadas a la información, Gothenburg, Suecia, 2010 [en línea]. Disponible en: http://publications.lib.chalmers.se/records/fulltext/138247.pdf. [consulta: 04 agosto 2016].

[7] M. Selinger y L. Schmidt. *Adaptive Traffic Control Systems in the United States*. Septiembre 2014 [en línea]. Disponible en: http://www.hdrinc.com/sites/all/files/assets/knowledge-center/articles/adaptive-traffic-control-systems-in-the-united-states-2010.pdf. [consulta: 04 agosto 2016].

[8] S. F. Smith, G. J. Barlow, X. Xie y Z.B. Rubinstein. "SURTRAC: Scalable Urban Traffic Control", *Transportation Research Board 92nd Annual Meeting Compendium of Papers*, enero 2013 [en línea]. Disponible en: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1877&context=robotics. [consulta: 04 agosto 2016].

[9] M. Wiering. *Multi-Agent Reinforcement Learning for Traffic Light Control*. Utrecht: Universidad de Utrecht, Departamento de informática, Utrecht, Países bajos, 2000 [en línea]. Disponible en: http://www.dcsc.tudelft.nl/~sc4081/assign/pap/Reinforcement_Learning.pdf. [consulta: 04 agosto 2016].

[10] M. Steingröver, R. Schouten, S. Peelen, E. Nijhuis y B. Bakker. *Reinforcement Learning of Traffic Light Controllers adapting to Traffic Congestion*. Ámsterdam: Universidad de Ámsterdam, Departamento de informática, Ámsterdam, Alemania, enero 2005 [en línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6290&rep=rep1&type=pdf. [consulta: 04 agosto 2016].

[11] Pattberg, B. *SUMO – Simulation of Urban MObility*. DLR Institute of Transportation Systems [en línea]. Disponible en: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/. [consulta: 4 agosto 2016].

[12] OpenStreetMap [en línea]. Disponible en: https://www.openstreetmap.org. [consulta: 4 agosto 2016].

[13] JOSM [en línea]. Disponible en: https://josm.openstreetmap.de. [consulta: 4 agosto 2016].

[14] TraCI/Protocol - Sumo [en línea]. Disponible en: http://sumo.dlr.de/wiki/TraCI/Protocol. [consulta: 4 agosto 2016].

[15] E. Even-Dar y Y. Mansour. "Learning Rates for Q-learning", *Journal of Machine Learning Research*, nº. 5, 1-25, diciembre 2003.

[16] Tools/Trip - Sumo [en línea]. Disponible en: http://sumo.dlr.de/wiki/Tools/Trip. [consulta: 4 agosto 2016].

[17] Iihs.org. Automated enforcement laws with map [en línea]. Disponible en: http://www.iihs.org/iihs/topics/laws/automated_enforcement [consulta: 4 agosto 2016].